OFS: An Overlay File System for Cloud-Assisted Mobile Applications

Jianchen Shan, **Nafize R. Paiker, Xiaoning Ding,**
Narain Gehani, Reza Curtmola, Cristian Borcea

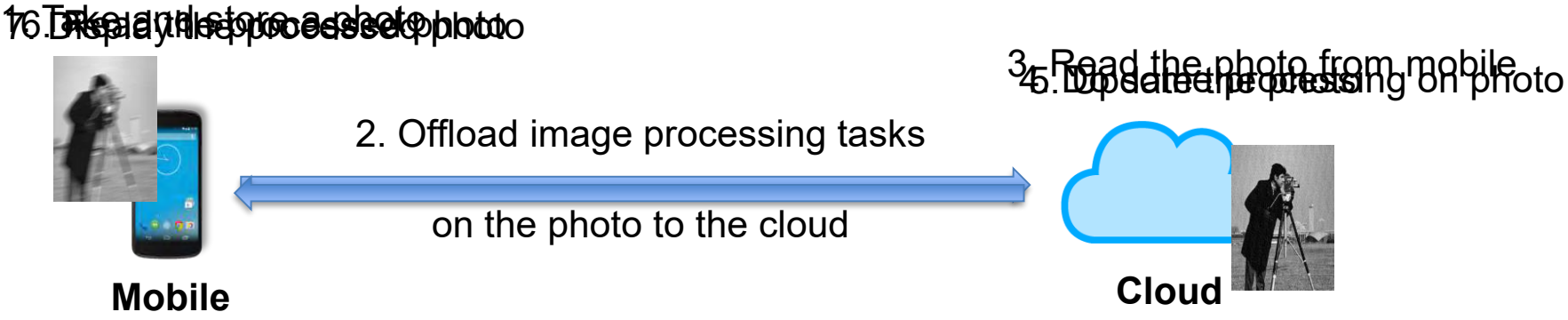
NJIT
New Jersey Institute of Technology

Mobile apps need cloud assistance

- **Mobile devices have limited resources**
- **Systems designed to offload resource-demanding tasks to cloud**
- **Tasks offloaded in the forms of:**
  - **Threads: CloneCloud [EuroSys ’11], COMET [OSDI ’12]**
  - **Procedures: MAUI [MobiSys ’10], ThinkAir [Infocom ’12]**
  - **Objects: Sapphire [OSDI ’14]**

Example of cloud-assisted mobile app

- **Enhanced camera app**



- **Characteristics of file I/O in cloud-assisted mobile apps:**
  - **Read and write files on both mobile and cloud**
  - **May require strong consistency (always read latest copy)**
  - **Long I/O latency due to transferring the file over network**

Existing systems cannot handle file I/O in cloud-assisted mobile apps

- **Don't support offloading tasks that perform file operations**
  - **COMET [OSDI '12]**
- **Don't have mechanisms to support consistent remote file access**
  - **MAUI [MobiSys '10], ThinkAir [Infocom '12], CloneCloud [EuroSys '11], Sapphire [OSDI '14]**

Problems with using network and distributed file systems

- **Strong consistency** cannot be achieved with low latency and low network overhead
- **Opened** files must be reopened after a task is offloaded in order to continue accessing the file
  - Close → migrate → reopen
- **Root privilege** needed to setup client software
- **User credentials** need to be saved in the cloud to access files

File system requirements for cloud-assisted mobile apps

- **Location transparency**
  - Access remote files as though they were local
  - Maintain file sessions during task offloading
- **Consistency**
  - Ensure correct execution of tasks distributed across mobile and cloud
- **Performance**
  - Provide low latency with little network overhead to save energy and network bandwidth
- **Ease of deployment**
  - Require minimal privileges in addition to those needed to run tasks
  - No need to save to-be-accessed files before application runs

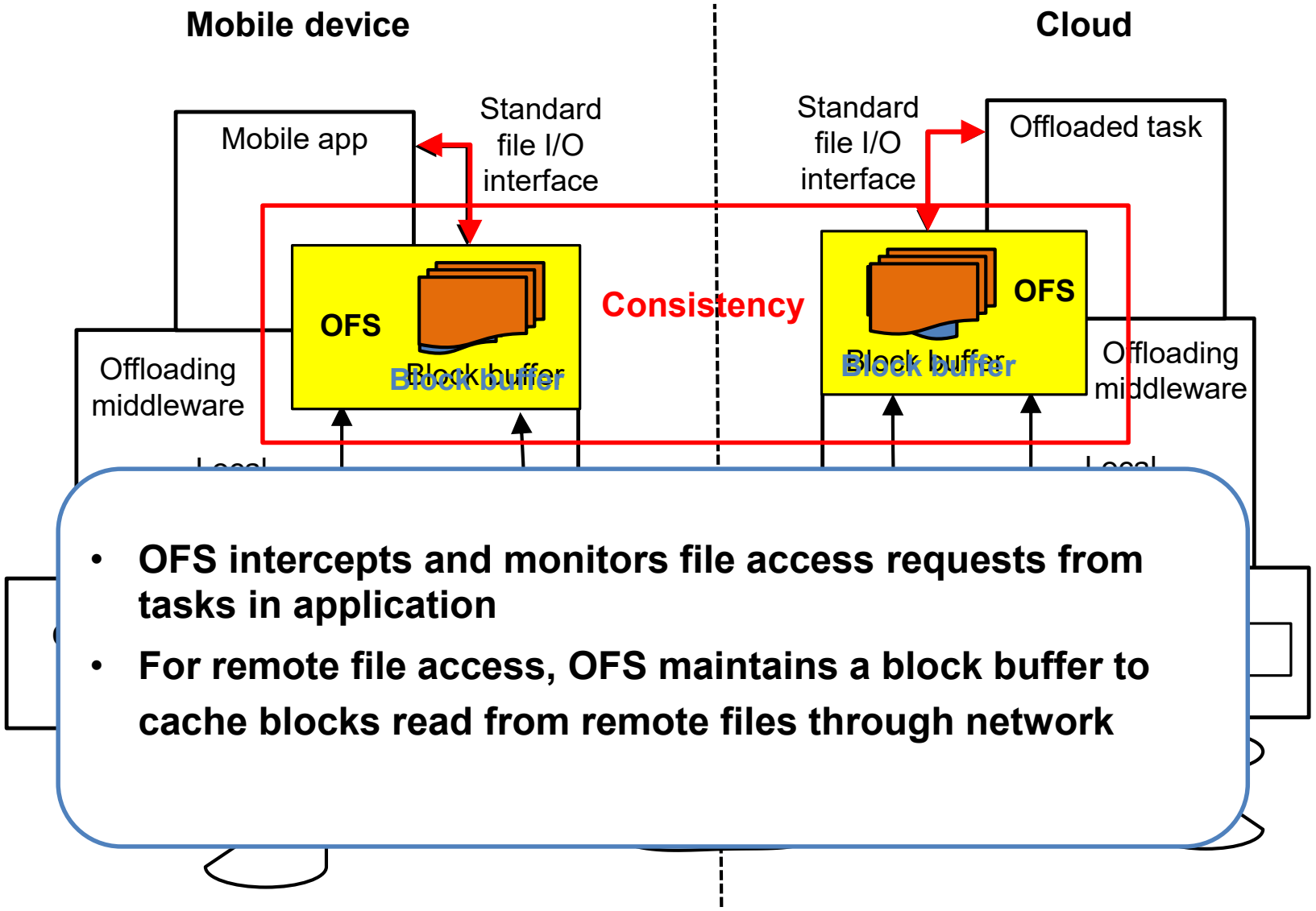
Overlay file system (OFS)

- Application-level **FS for cloud assisted mobile apps**
  - **Doesn't need system-wide management**
  - **Can work with any native file system**
  - **Doesn't incur costly context switches**
- **Advantages:**
  - **Strong consistency (delayed-update policy)**
  - **Location transparency (file session management)**
  - **Low overhead (low latency file access, low overhead consistency maintenance)**
  - **Ease of deployment (application level)**

Outline

- Background
- OFS
  - Architecture and design
  - Consistency model
- Evaluation
- Conclusion and future work

Overall system architecture

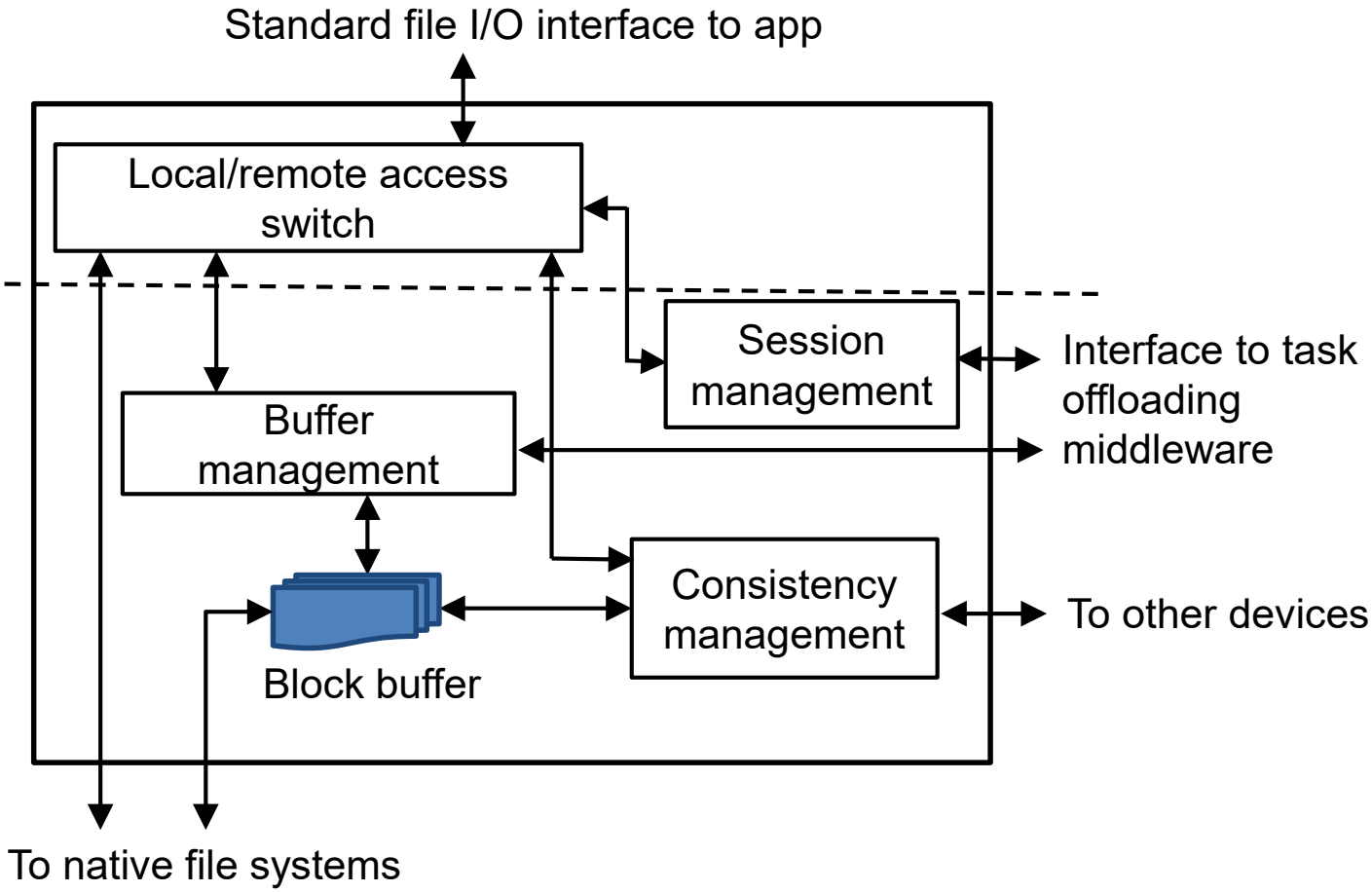


- **OFS intercepts and monitors file access requests from tasks in application**
- **For remote file access, OFS maintains a block buffer to cache blocks read from remote files through network**

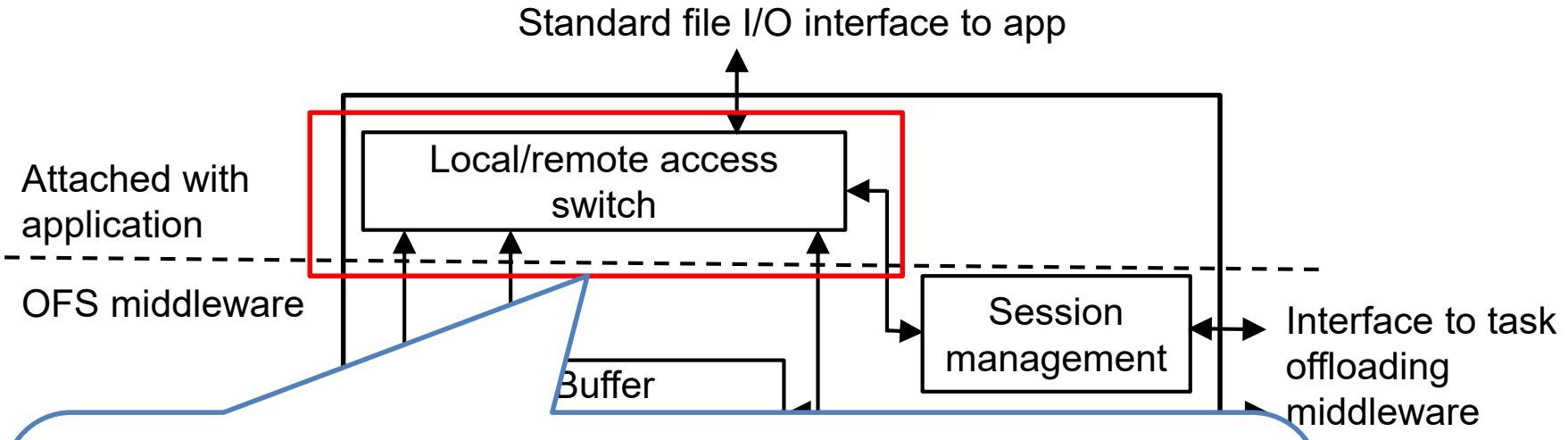
OFS architecture

Upper layer:
library linked with application

Lower layer:
app-level runtime service in middleware

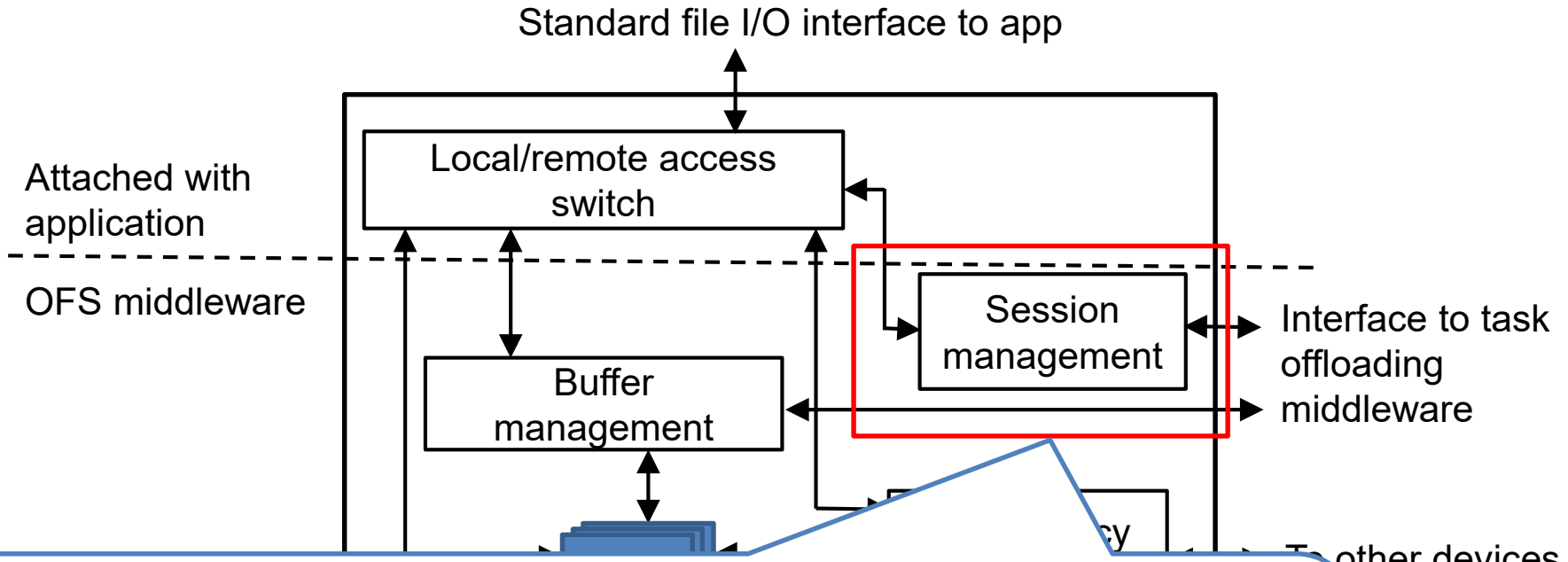


OFS architecture



- **Intercepts library calls**
- **Decides whether a call should be handled by native file system or OFS**
  - **Native file system: local files**
  - **OFS: remote files**

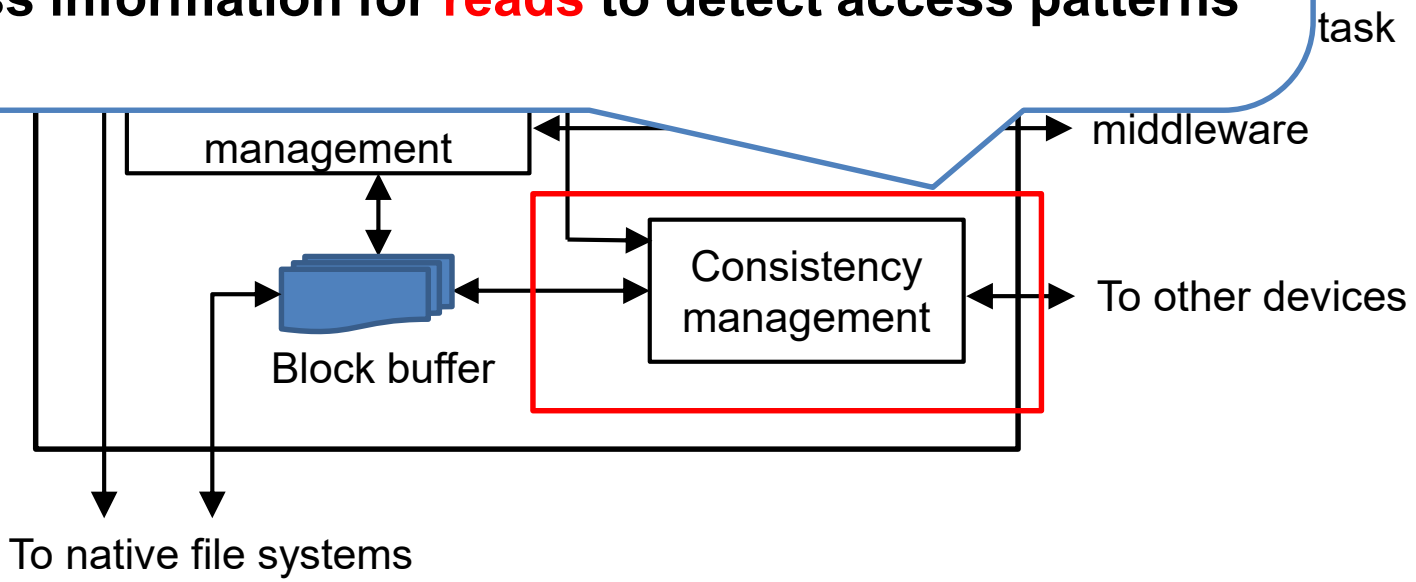
OFS architecture



- **File session: set of file operations and states between file open and close**
- **When a task is offloaded, the state required by the unfinished file sessions will be correctly set up**

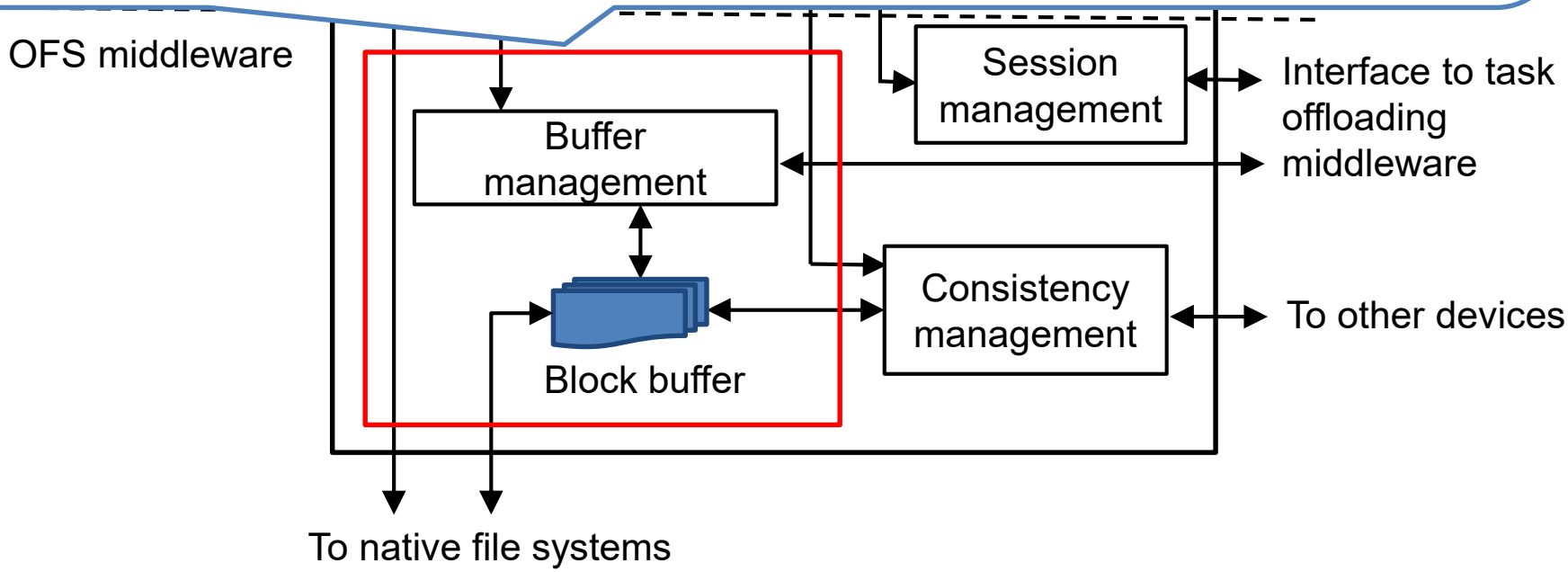
OFS architecture

- **Utilizes the delay-update consistency model**
- **Notified of all calls before it passes the calls to buffer management**
- **Confirms that writes will not cause inconsistency issues**
- **Keeps access information for reads to detect access patterns**

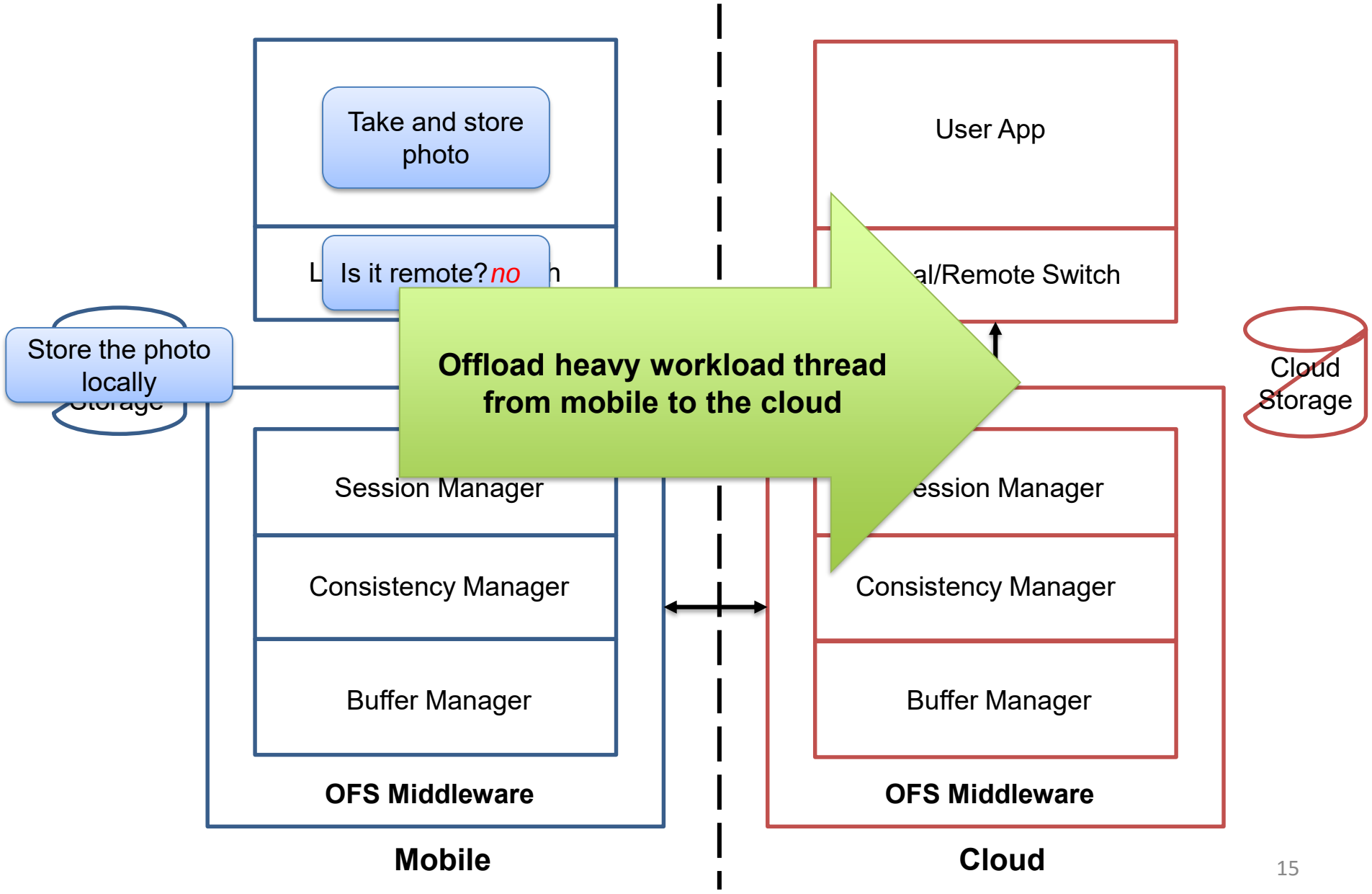


OFS architecture

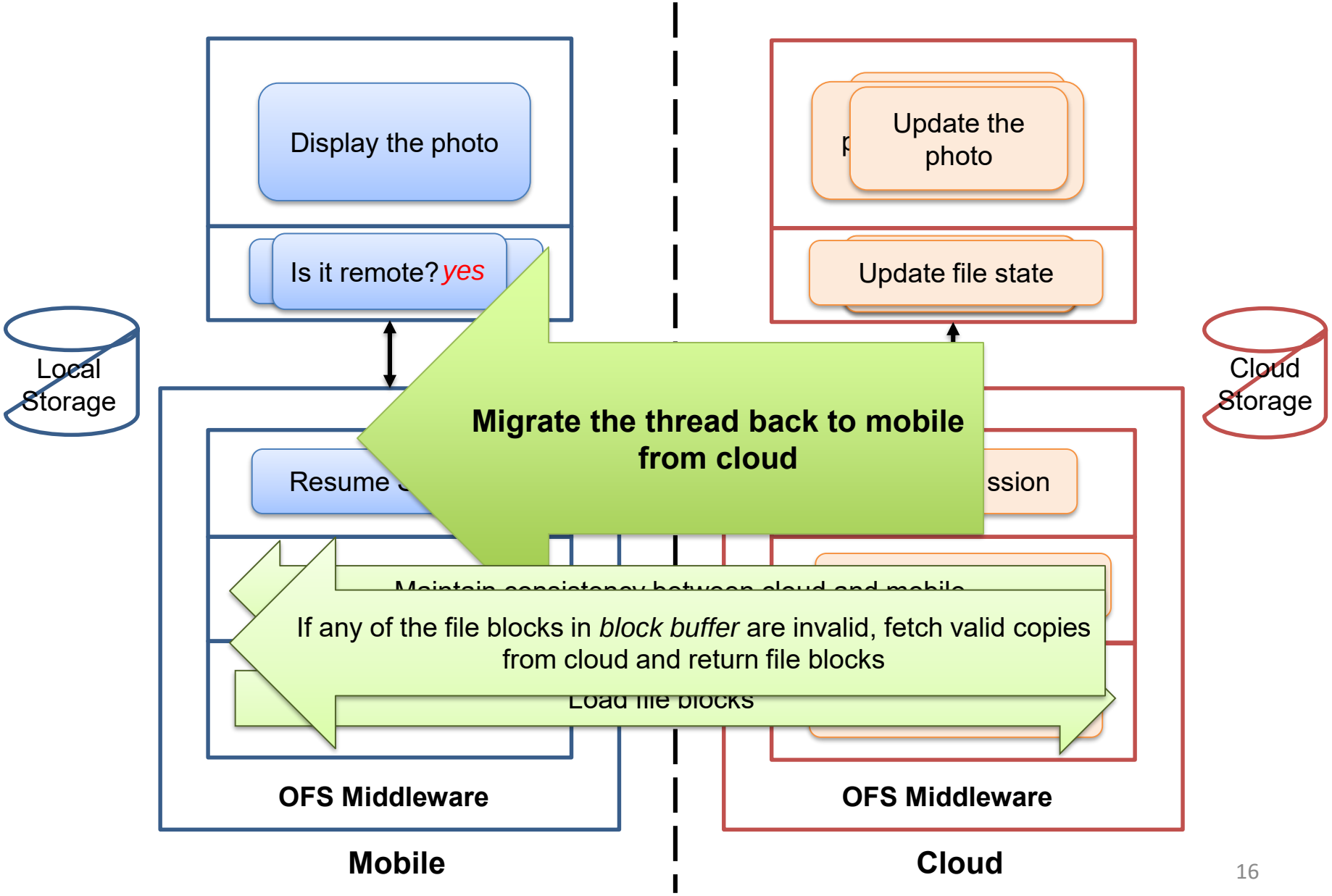
- **Caches only remote files**
- **Data and metadata reside in virtual address space for fast access**
- **Metadata maintains location and status of file blocks**



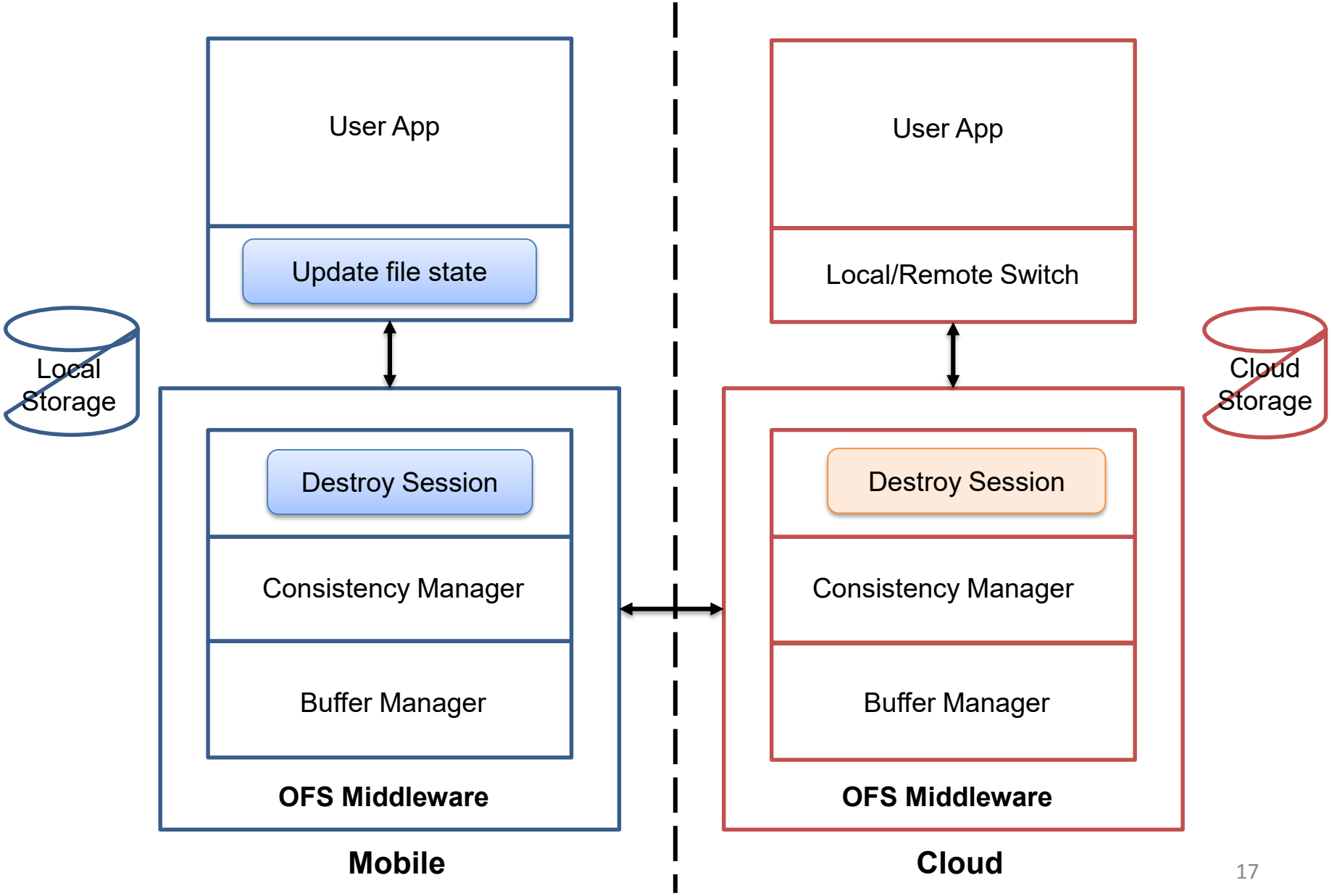
OFS workflow: enhanced camera app



OFS workflow: enhanced camera app



OFS workflow: enhanced camera app

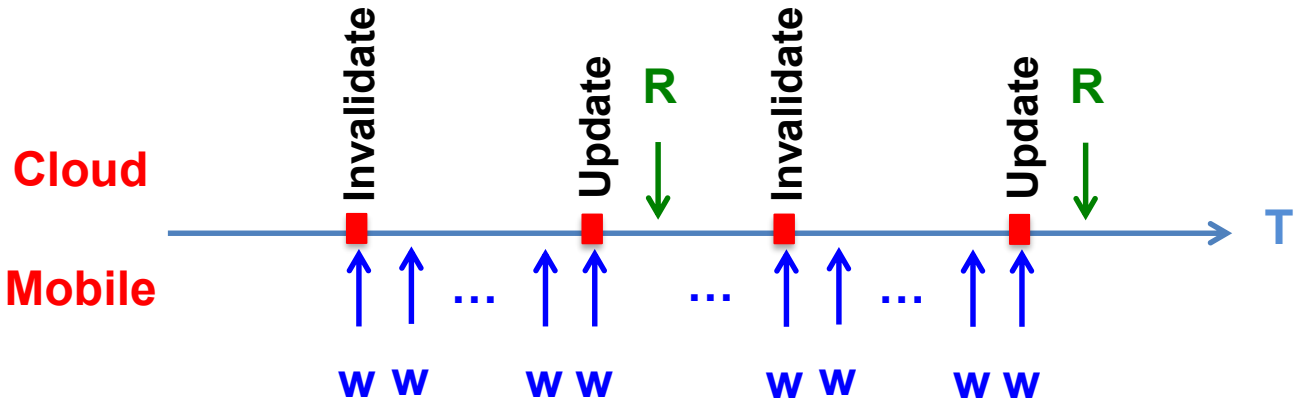


Outline

- Background
- **OFS**
  - Architecture and design
  - **Consistency model**
- **Evaluation**
- **Conclusion and future work**

Delayed-update algorithm

- **Monitor file access pattern to efficiently maintain strong consistency**
  - **Combination of write-invalidate and write-update**
    - **Invalidate duplicates**
    - **Update them when they are about to be read**



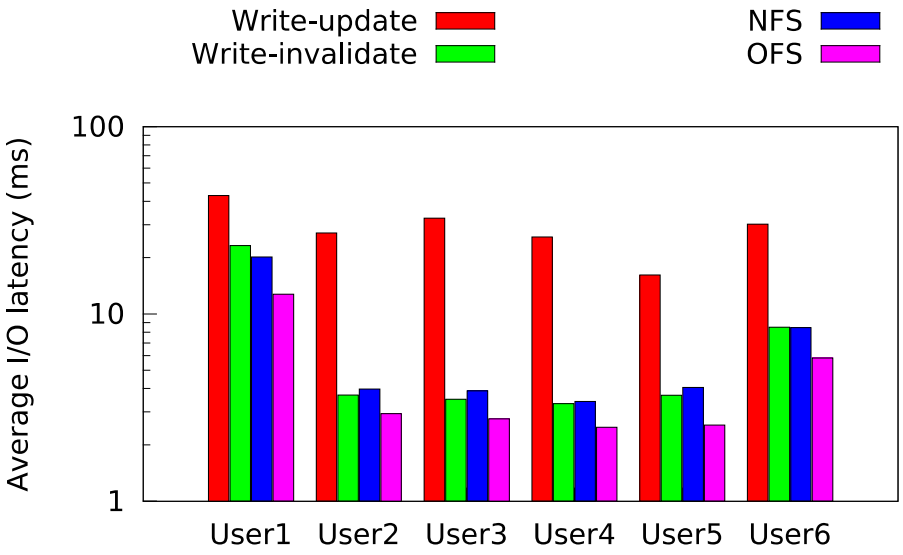
Outline

- Background
- OFS
  - Architecture and design
  - Consistency model
- **Evaluation**
- **Conclusion and future work**

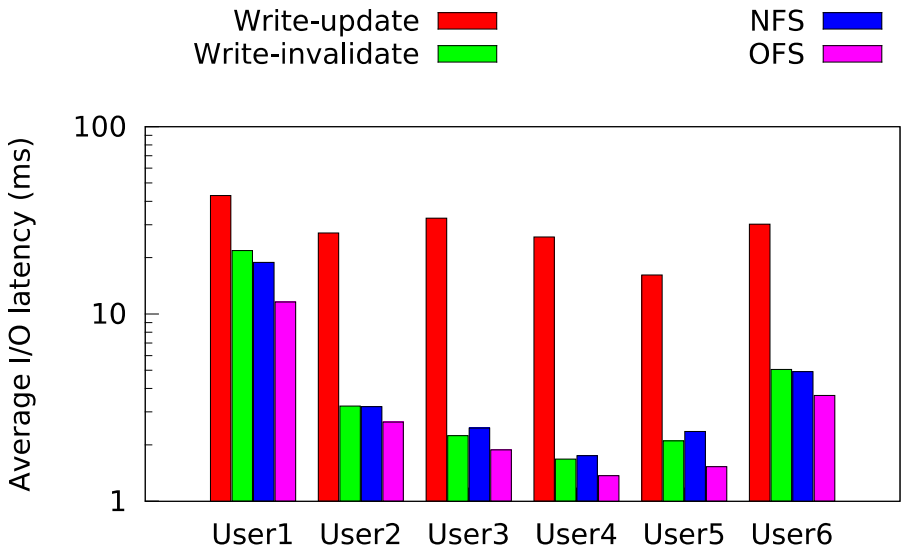
Experiments

- **Goal: compare OFS against with write-invalidate, write-update, and NFS**
- **Traces: real-life mobile app file access traces derived from U. of Buffalo's PhoneLab testbed**
  - **Thread offloading: offload complete threads**
  - **Procedure offloading: offload parts of threads**
- **Metrics:**
  - **Average read and write latency**
  - **Average I/O latency**
  - **Network overhead**
  - **Mobile device active time**

OFS incurs lowest I/O latency



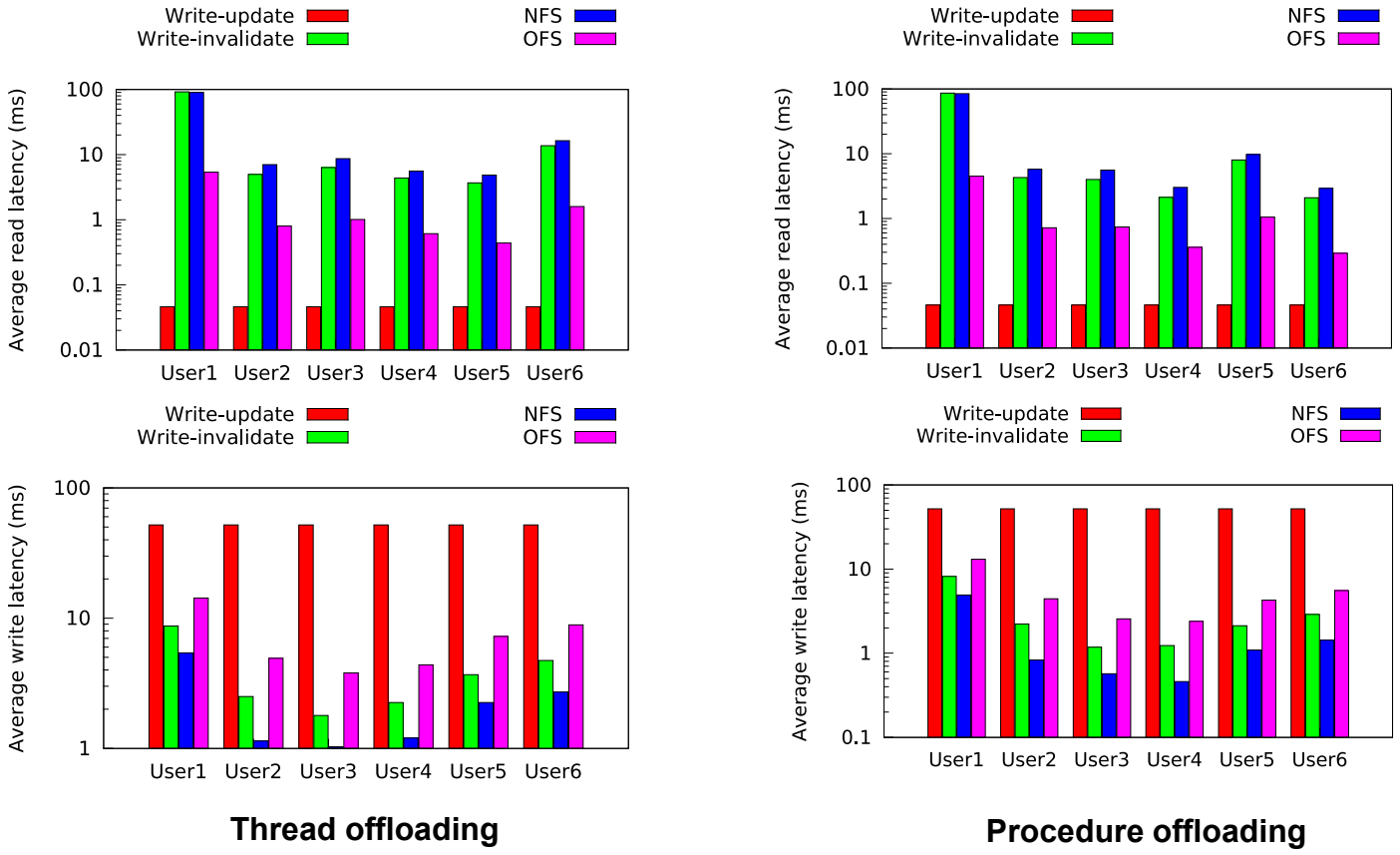
Thread offloading



Procedure offloading

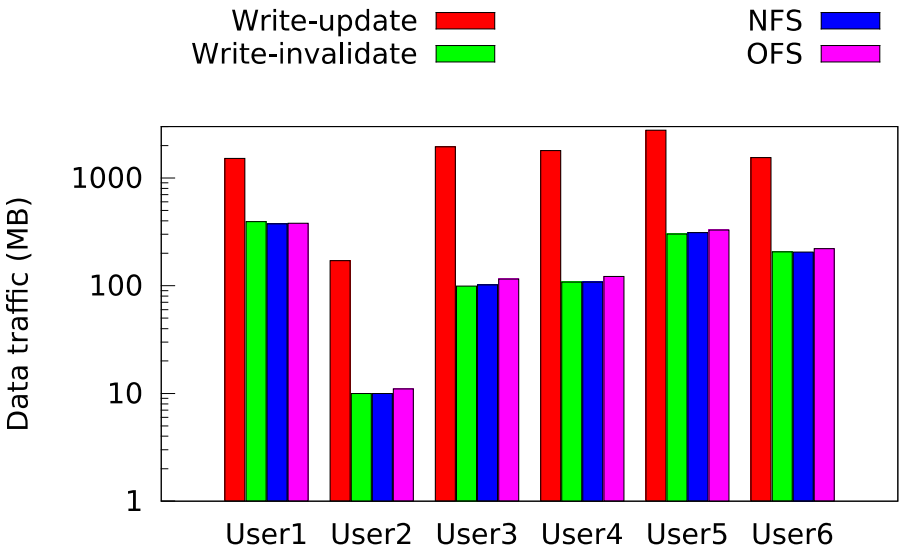
- **85%, 37%, and 33% lower latency than write-update, write-invalidate, and NFS**
- **Procedure offloading incurs 22% lower latency than thread offloading**

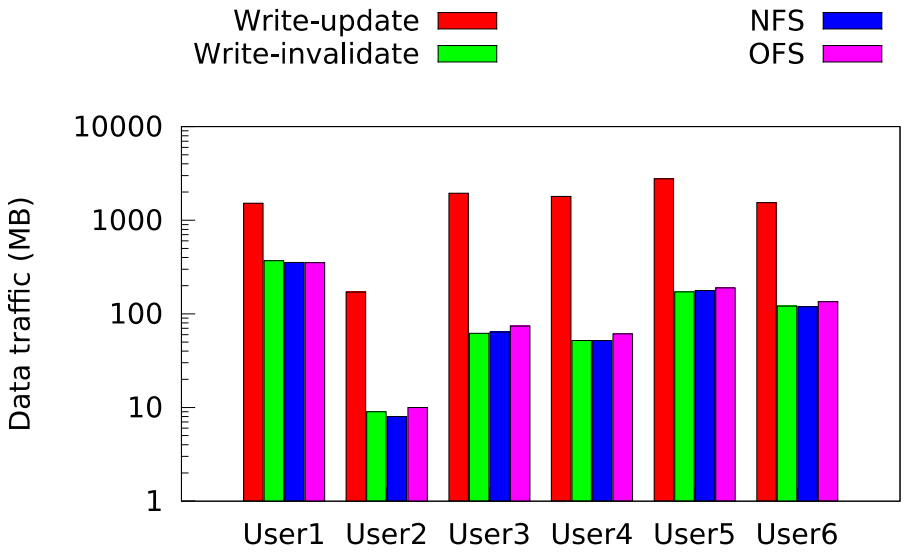
OFS reduces read latency at the cost of write latency



- **Read latency is 14x lower, and write latency is 2-3x higher than write invalidate and NFS**
- **Read-intensive workloads benefit more from OFS than write-intensive workloads**

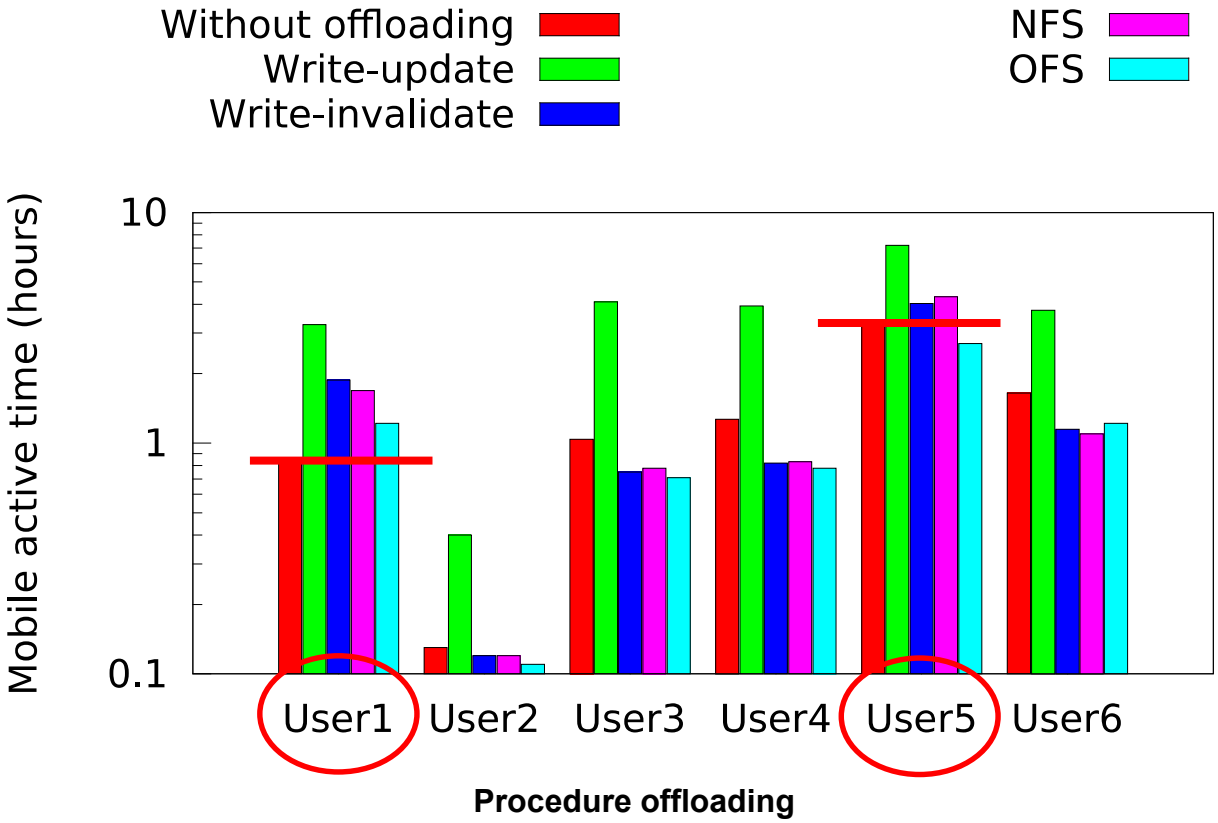
OFS achieves lower I/O latency with slightly higher network overhead





- **6% higher network overhead than write-invalidate and NFS**
- **Procedure offloading leads to lower network overhead than thread offloading**

OFS reduces mobile device active time more effectively than other policies



- **I/O overhead must be effectively reduced to really benefit from task-offloading to the cloud**
  - **Speedup application and save battery power**

Conclusion and future work

- **OFS provides efficiency, consistency, and location transparency**
- **OFS lowers substantially file access latency at the cost of small network overhead**
- **OFS reduces the active time of mobile devices when running cloud-assisted apps**
- **OFS is more effective with read-intensive workloads and procedure offloading**
- **Future work: integrating OFS in our *Moitree* middleware**

Thanks!

http://cs.njit.edu/~borcea/avatar

Acknowledgment: NSF Grants CNS 1409523, CNS 1054754, DGE 1565478, and DUE 1241976. NSA Grant H98230-15-1-0274, DARPA/AFRL Contract: A8650-15-C-7521.

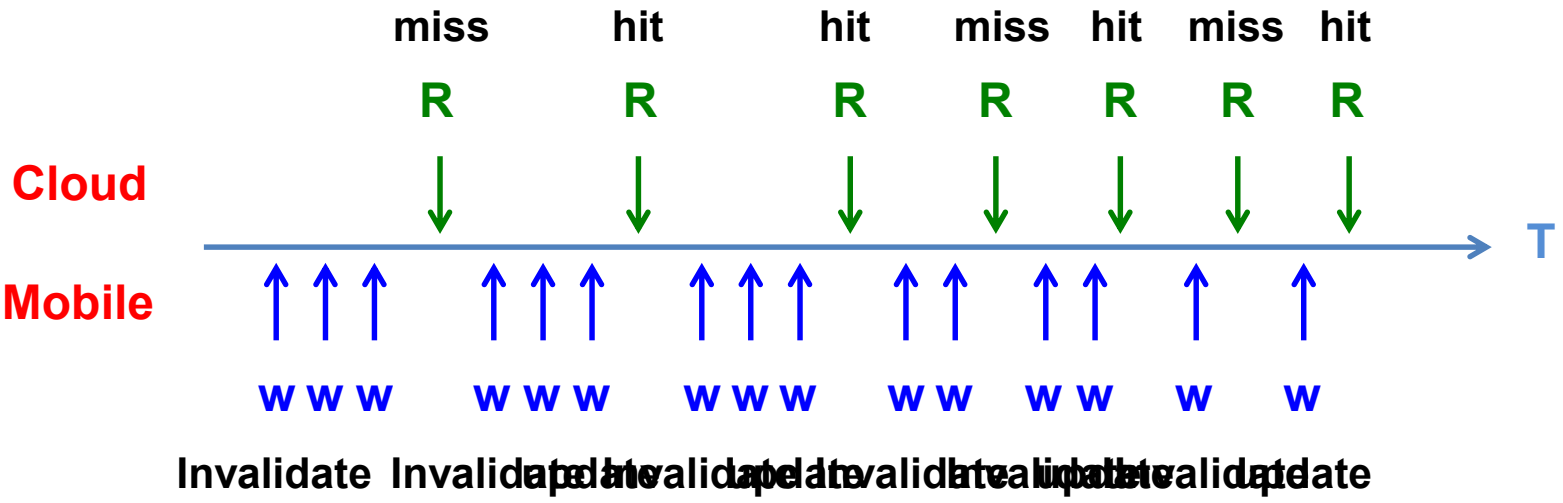
Backup slides

Consistency management in OFS

- **Objectives:**
  - **Strong consistency**
    - **No stale data => application correct execution, simple application development**
  - **Low access latency and network overhead**
    - **Write-invalidate and write-update**
  - **Relaxed consistency**
    - **Health monitoring app**

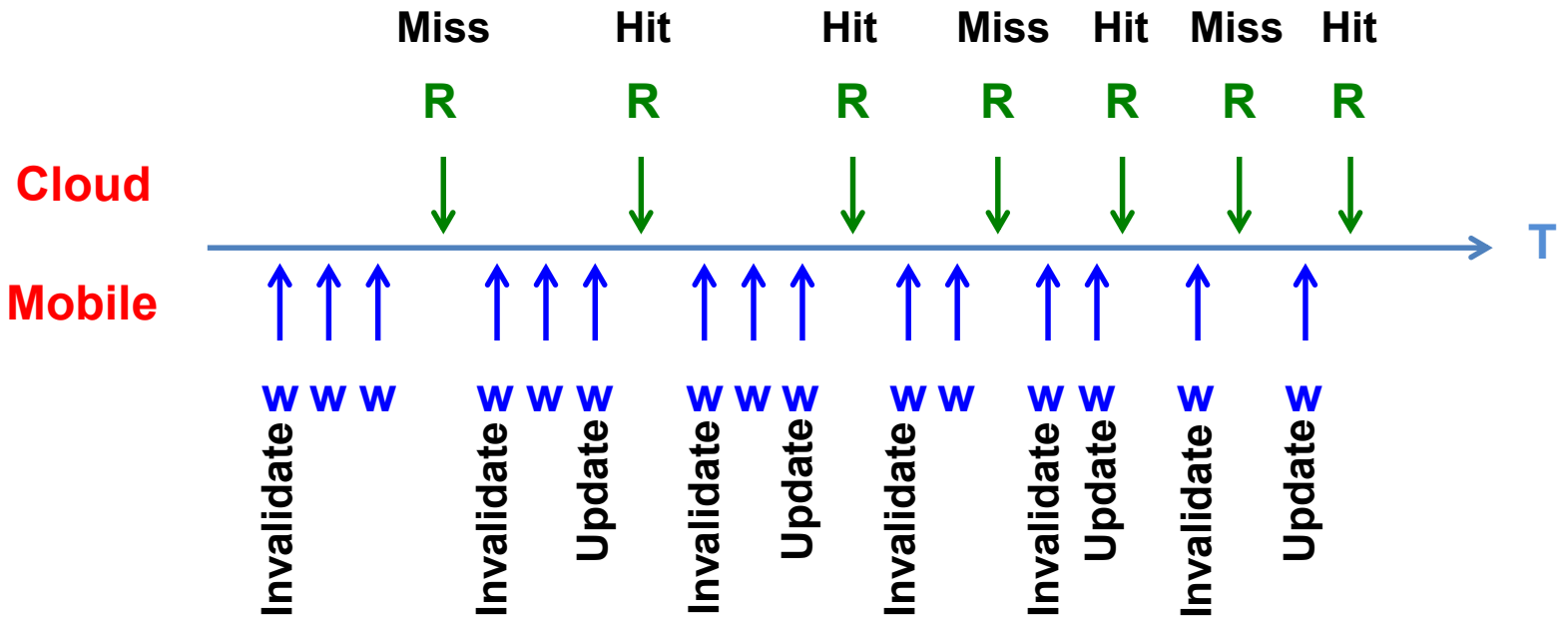
Delayed-update algorithm

- **Monitor file access pattern to determine when to update duplicates**
  - **Combination of write-invalidate and write-update**
  - **Invalidate duplicates and then update them when they are about be to read**

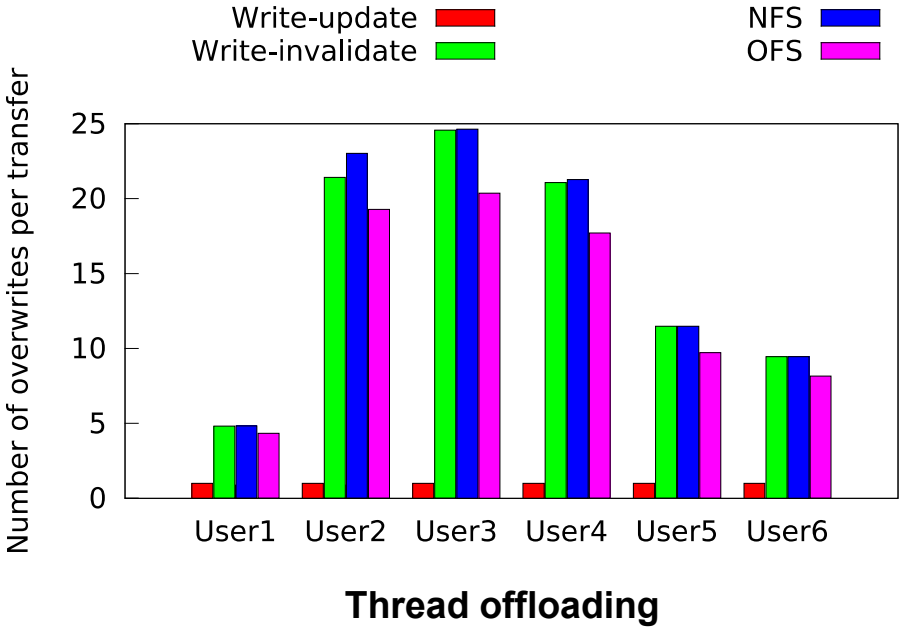


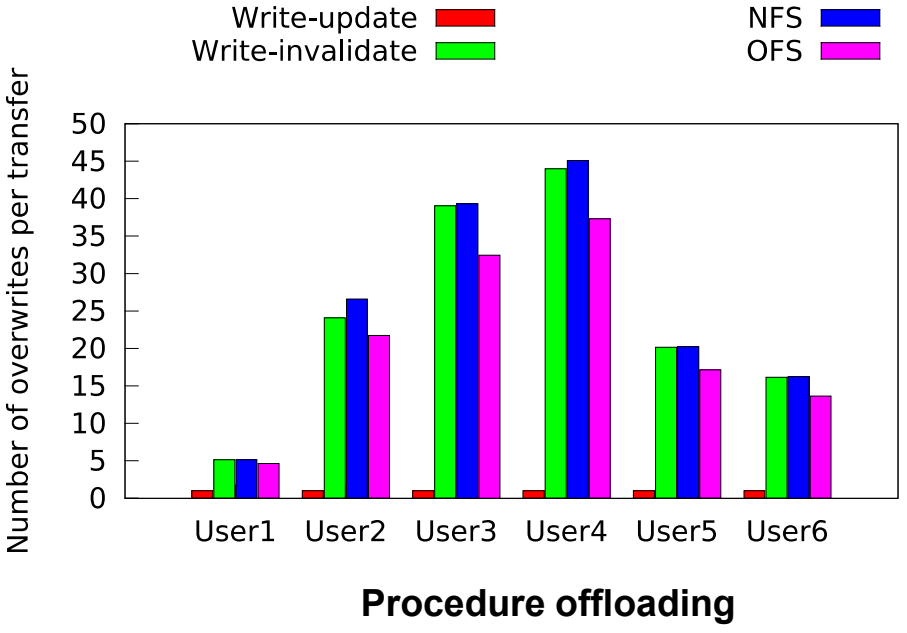
Delayed-update algorithm

- **Monitor file access pattern to efficiently maintain strong consistency**
  - **Combination of write-invalidate and write-update**
  - **Invalidate duplicates; update them when they are about be to read**



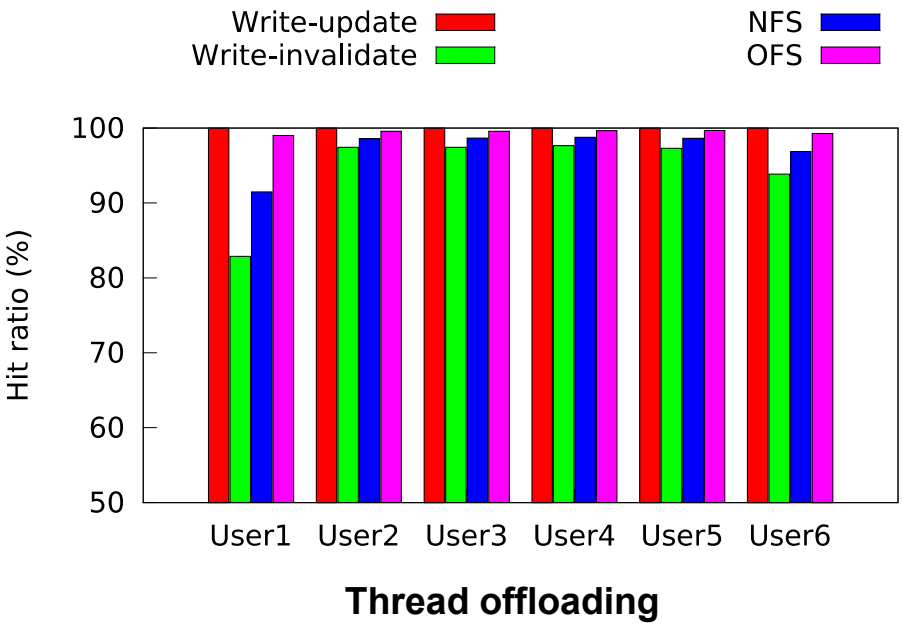
Number of overwrites per transfer



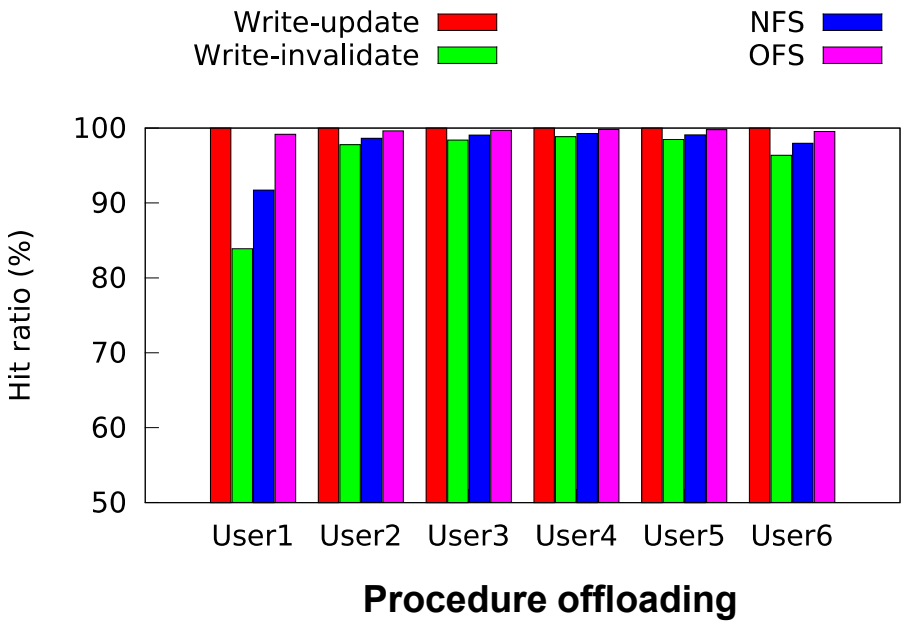


- **OFS transfer data more frequently than write-invalidate and NFS**
  - **Cannot accurately predict and may update data too soon**
  - **Explain why OFS incurs slightly higher write latency and network overhead**
- **OFS transfer data less frequently for procedure offloading**
  - **Explain why procedure offloading perform better than thread offloading in terms of write latency and network overhead**

Block buffer hit ratio



Thread offloading



Procedure offloading

- **OFS has higher hit ratio than write-invalidate and NFS (99% Avg)**
  - **Indicates that the delayed-update algorithm in OFS can effectively adjust the overwrite threshold and update duplicates to minimize buffer misses**
  - **Explains why OFS has lower read latency**
- **Procedure offloading has higher hit ratio than thread offloading**
  - **Explains why the read latency is lower for procedure offloading than for thread offloading**