Scale and Performance in a Filesystem Semi-Microkernel

李缙、徐宇鸣

Outline

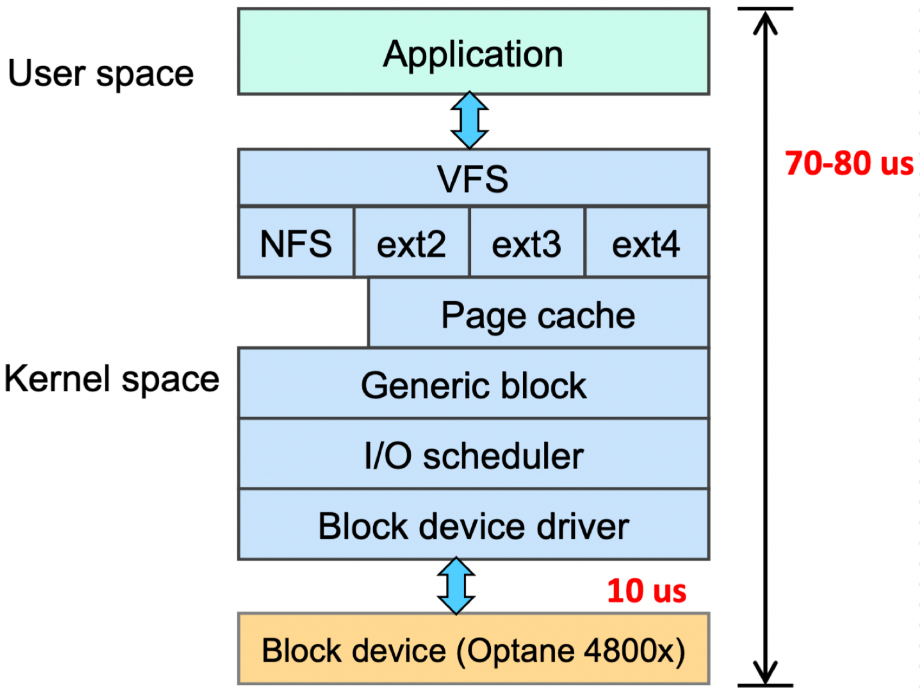
- **Background**
- **uFS Design**
- **Evaluation**
- **Conclusion**

Outline

- **Background**
- uFS Design
- Evaluation
- Conclusion

Background

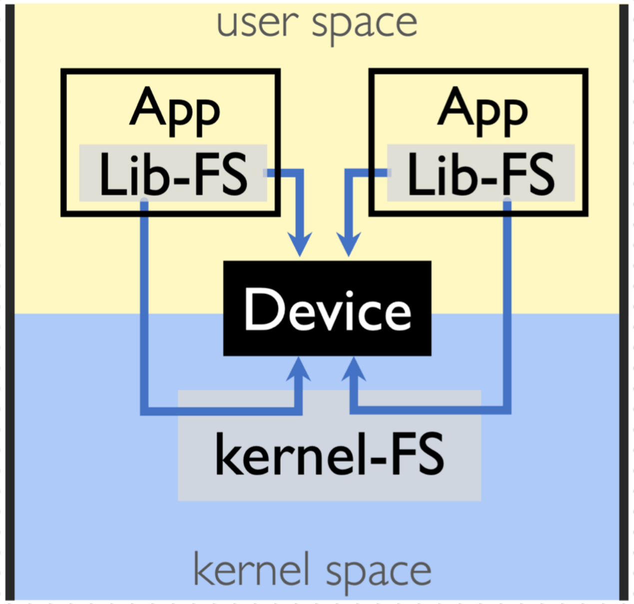
- **HW is Fast – but SW Appears Slow**
  - **notable overhead to trapping in and out of the kernel**
    - **Ep: For Optane 4800x, device cost ~10us, Linux I/O stack cost ~70us**



Linux I/O Stack

Background

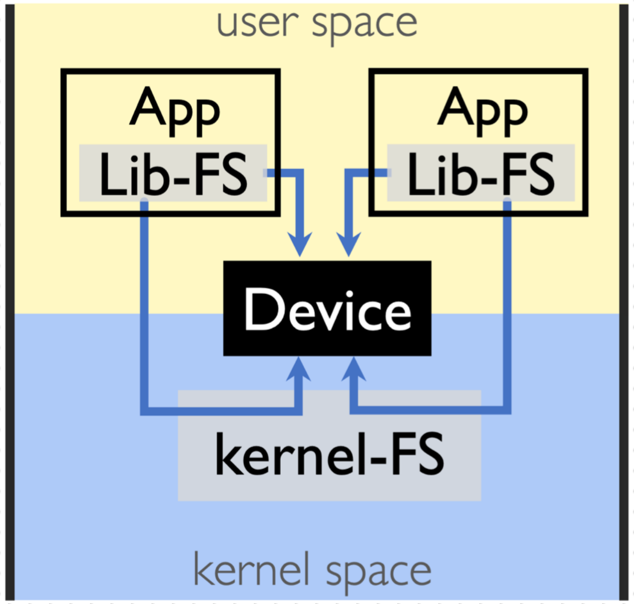
- **HW is Fast – but SW Appears Slow**
  - **notable overhead to trapping in and out of the kernel**
    - **Ep: For Optane 4800x, device cost ~10us, Linux I/O stack cost ~70us**
- **Existing Solutions:**
  - **Libraries directly access the device**
    - **Strata(SOSP-17), SplitFS(SOSP-19)**



Background

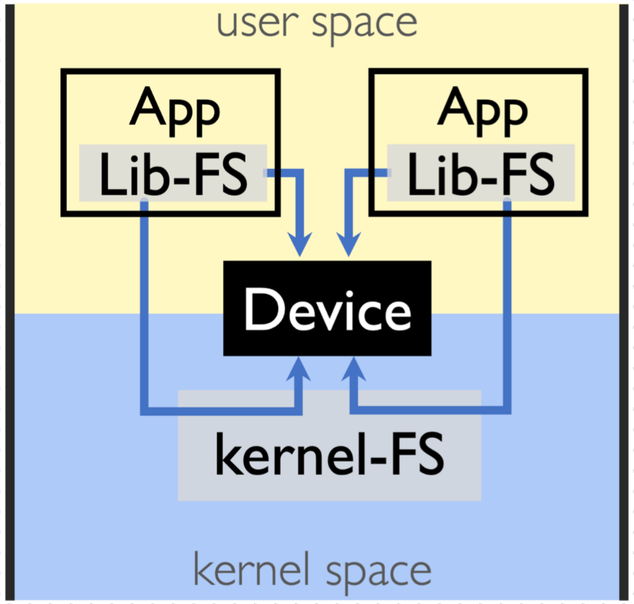
USTC, CHINA ADSLAB

- **HW is Fast – but SW Appears Slow**
  - **notable overhead to trapping in and out of the kernel**
    - **Ep: For Optane 4800x, device cost ~10us, Linux I/O stack cost ~70us**
- **Existing Solutions:**
  - **Libraries directly access the device**
    - **Strata(SOSP-17), SplitFS(SOSP-19)**
  - **Drawback:**
    - **Complicate the device access isolation and sharing**



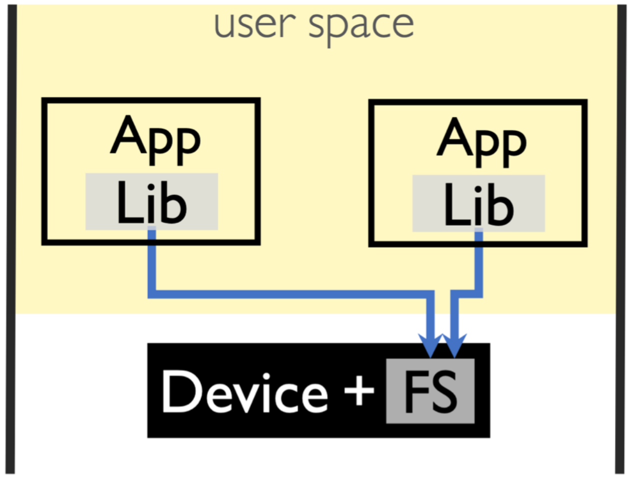
Background

- **HW is Fast – but SW Appears Slow**
  - **notable overhead to trapping in and out of the kernel**
    - **Ep: For Optane 4800x, device cost ~10us, Linux I/O stack cost ~70us**
- **Existing Solutions:**
  - **Libraries directly access the device**
    - **Strata(SOSP-17), SplitFS(SOSP-19)**
  - **Drawback:**
    - **Complicate the device access isolation and sharing**
  - **Conclusion**
    - **Centralized IO multiplexing**
    - **Simpler isolation and sharing**



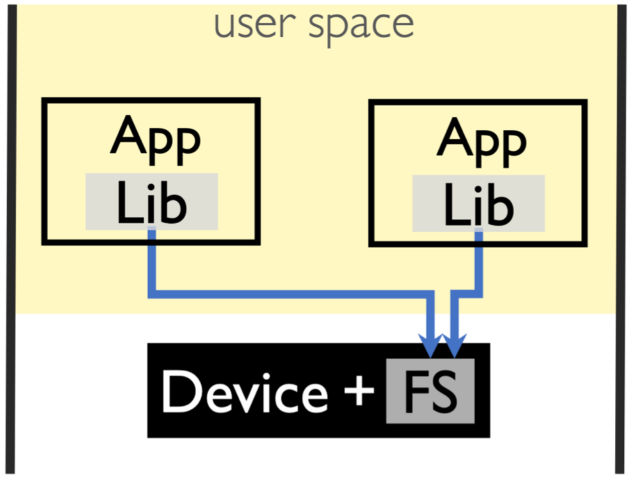
Background

- **HW is Fast – but SW Appears Slow**
  - **notable overhead to trapping in and out of the kernel**
    - **Ep: For Optane 4800x, device cost ~10us, Linux I/O stack cost ~70us**
- **Existing Solutions:**
  - **Move Filesystems to the device**
    - **DevFS(FAST-18), CrossFS(OSDI-20)**



Background

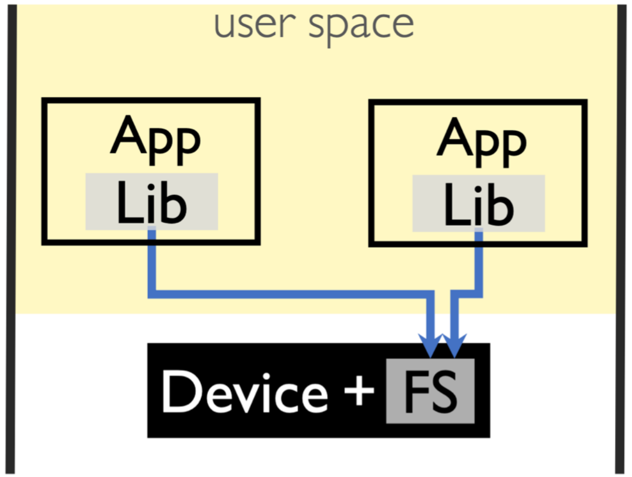
- **HW is Fast – but SW Appears Slow**
  - **notable overhead to trapping in and out of the kernel**
    - **Ep: For Optane 4800x, device cost ~10us, Linux I/O stack cost ~70us**
- **Existing Solutions:**
  - **Move Filesystems to the device**
    - **DevFS(FAST-18), CrossFS(OSDI-20)**
  - **Drawback:**
    - **“Smarter-HW” assumption**
    - **Unknown HW constraints**



Background

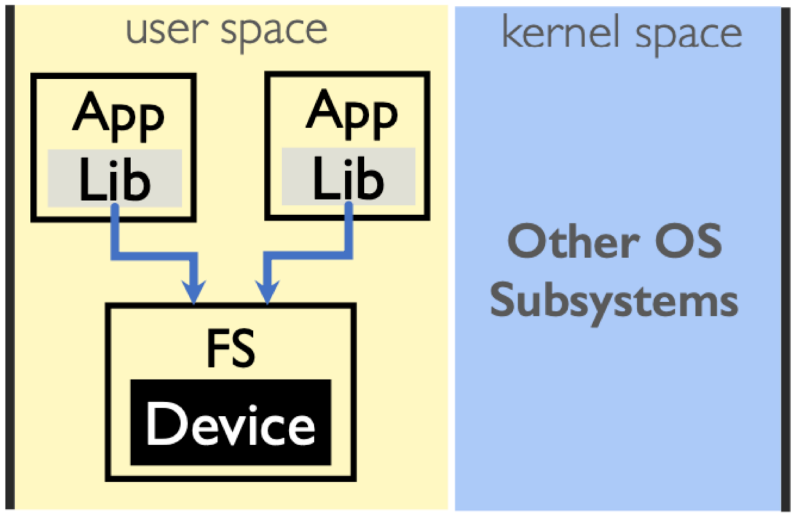
USTC, CHINA ADSLAB

- HW is Fast – but SW Appears Slow
  - notable overhead to trapping in and out of the kernel
    - Ep: For Optane 4800x, device cost ~10us, Linux I/O stack cost ~70us
- Existing Solutions:
  - Move Filesystems to the device
    - DevFS(FAST-18), CrossFS(OSDI-20)
  - Drawback:
    - "Smarter-HW" assumption
    - Unknown HW constraints
  - Conclusion
    - Realistic Assumption
    - Ultra-fast Devices and NVMe protocol



Background

- **HW is Fast – but SW Appears Slow**
  - **notable overhead to trapping in and out of the kernel**
    - **Ep: For Optane 4800x, device cost ~10us, Linux I/O stack cost ~70us**
- **Possible solution:**
  - **Semi-Microkernel**
    - **Or “filesystem as a process”(HotStorage-19)**



Background

USTC, CHINA ADSLAB

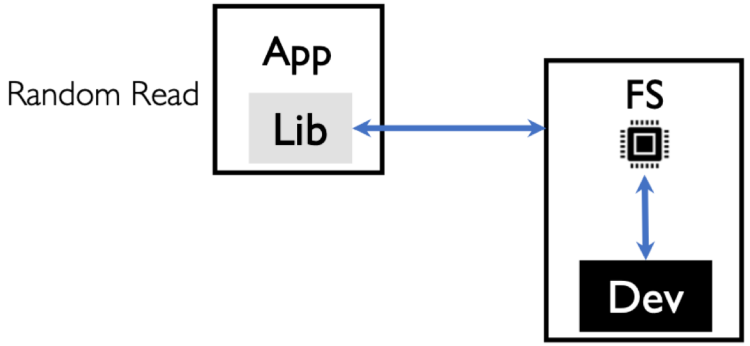
- **Semi-Microkernel**
  - **An OS subsystem that runs as a user-level process**
  - **Works in tandem with monolithic kernel**
- **Benefits of Semi-Microkernel**
  - **Code velocity**
    - **Quickly develop, modify, and deploy system software**
    - **Application-level debugging and testing**
  - **Performance**
    - **Scale subsystem independently from applications**
    - **Avoid extra kernel overhead**

Background

- Semi-Microkernel
  - An OS subsystem that runs as a user-level process
  - Works in tandem with monolithic kernel
- Prior semi-microkernel
  - Focus on networking
    - Snap(SOSP-19), TAS(Eurosys-19)
- Possible for storage now
  - User-level device driver(SPDK)

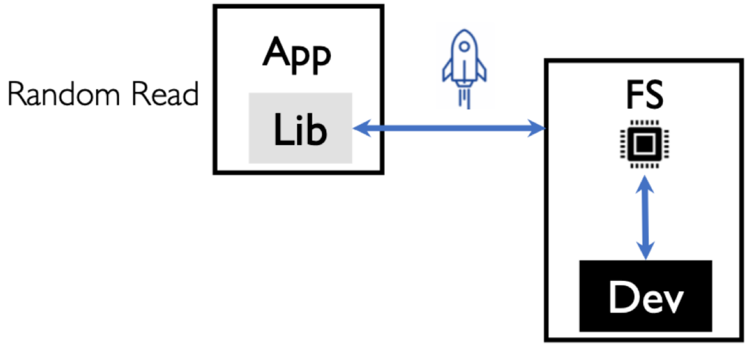
Background

- **Challenge**
  - **Base Performance**
    - **Inter-process communication & device access**



Background

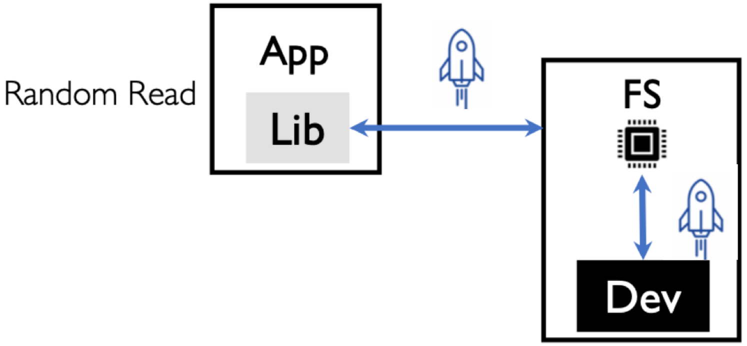
- Challenge
  - Base Performance
    - Inter-process communication & device access



Background

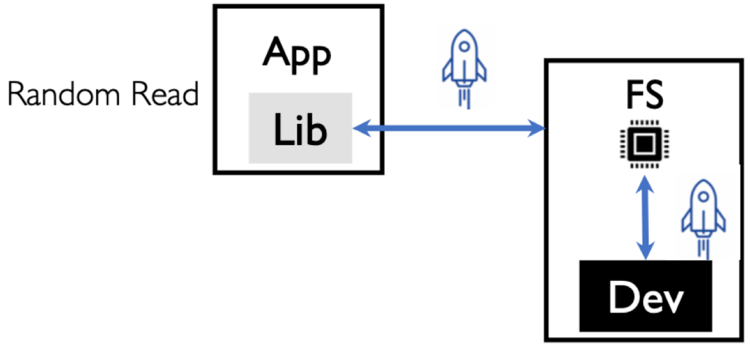
USTC, CHINA ADSLAB

- Challenge
  - Base Performance
    - Inter-process communication & device access



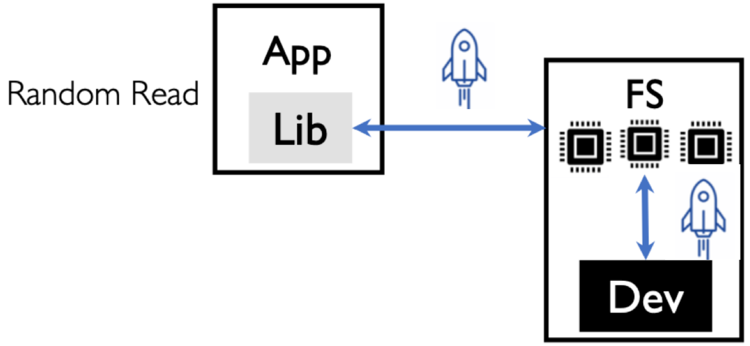
Background

- **Challenge**
  - **Base Performance**
    - **Inter-process communication & device access**
  - **Scale up and down**
    - **Scalability**



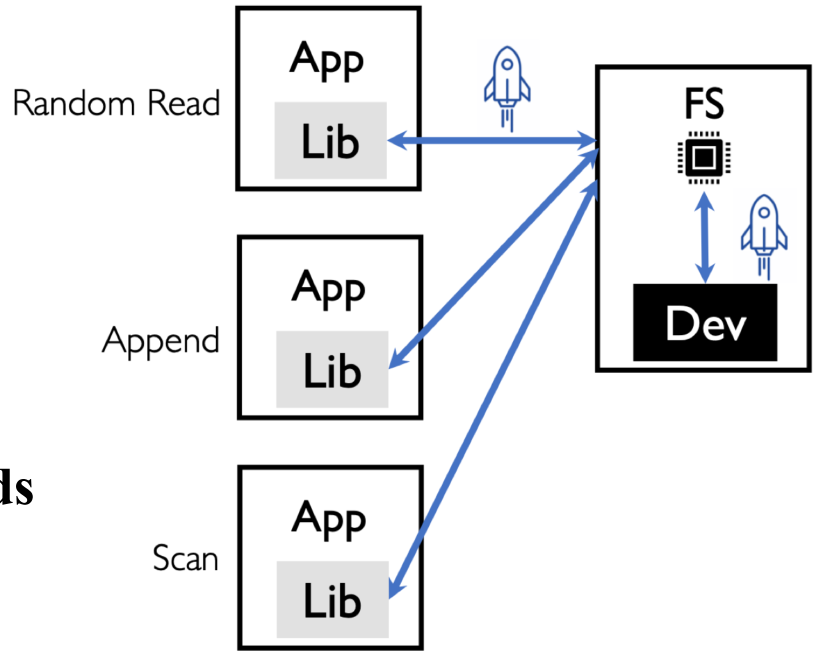
Background

- **Challenge**
  - **Base Performance**
    - **Inter-process communication & device access**
  - **Scale up and down**
    - **Scalability**



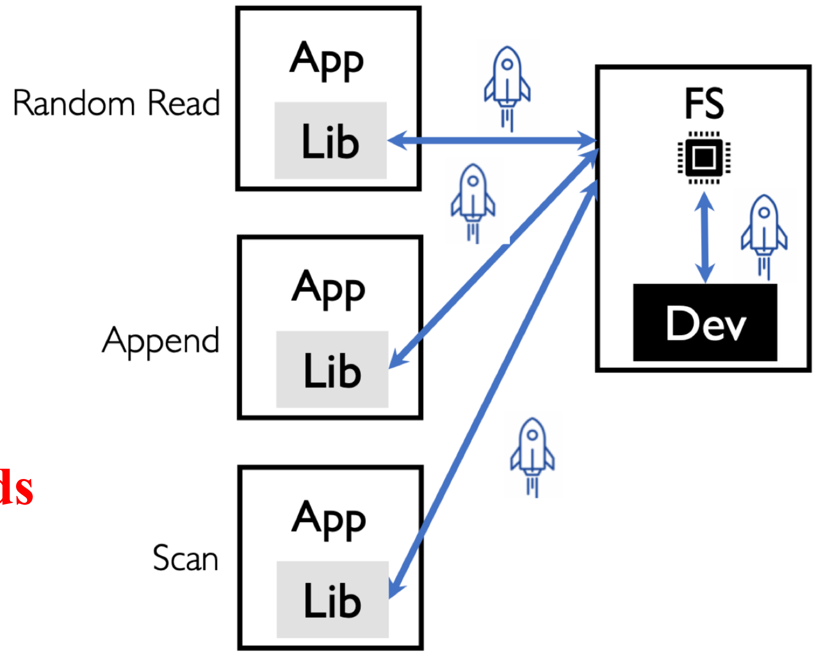
Background

- **Challenge**
  - **Base Performance**
    - **Inter-process communication & device access**
  - **Scale up and down**
    - **Scalability**
    - **Dynamic and heterogeneous application demands**



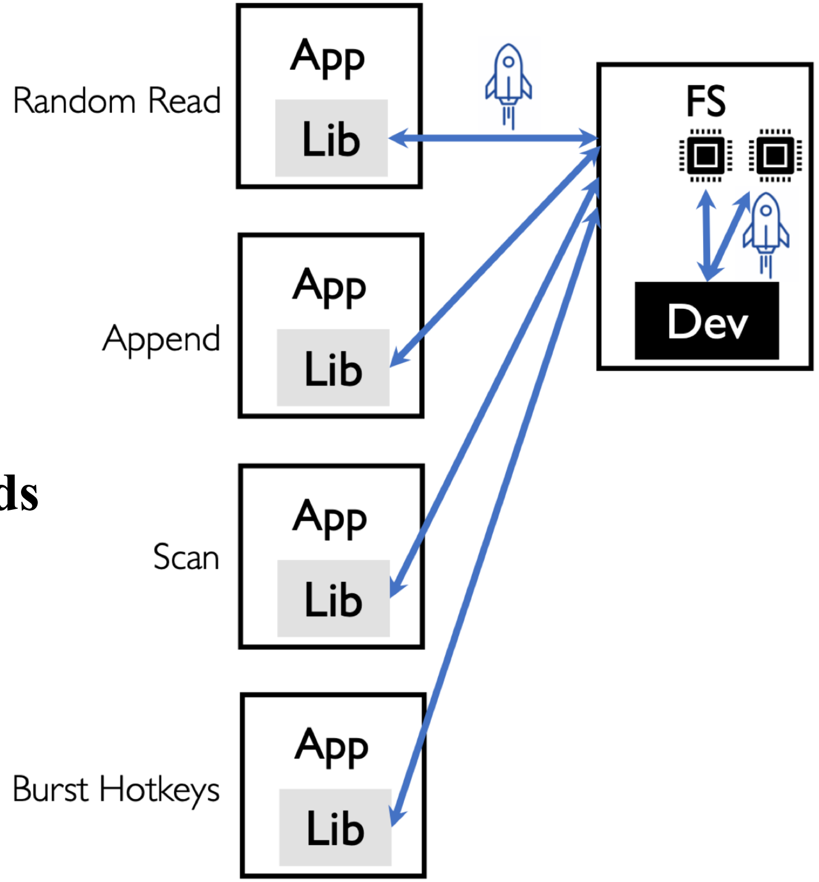
Background

- **Challenge**
  - **Base Performance**
    - **Inter-process communication & device access**
  - **Scale up and down**
    - **Scalability**
    - **Dynamic and heterogeneous application demands**



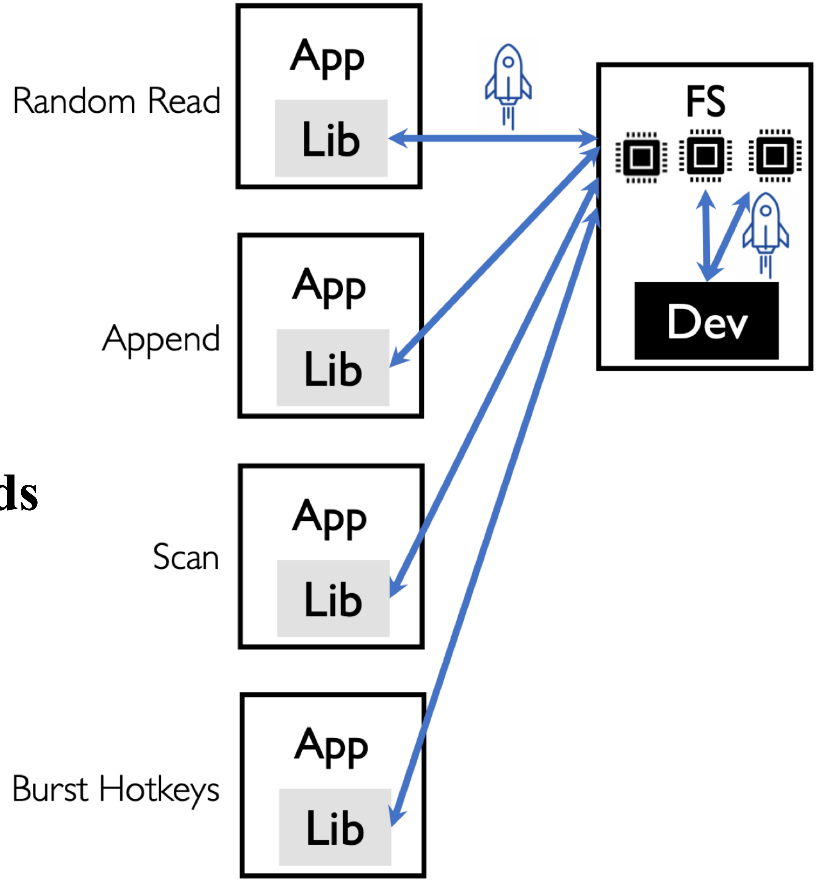
Background

- **Challenge**
  - **Base Performance**
    - **Inter-process communication & device access**
  - **Scale up and down**
    - **Scalability**
    - **Dynamic and heterogeneous application demands**
    - **Invest just-right amount of CPU**



Background

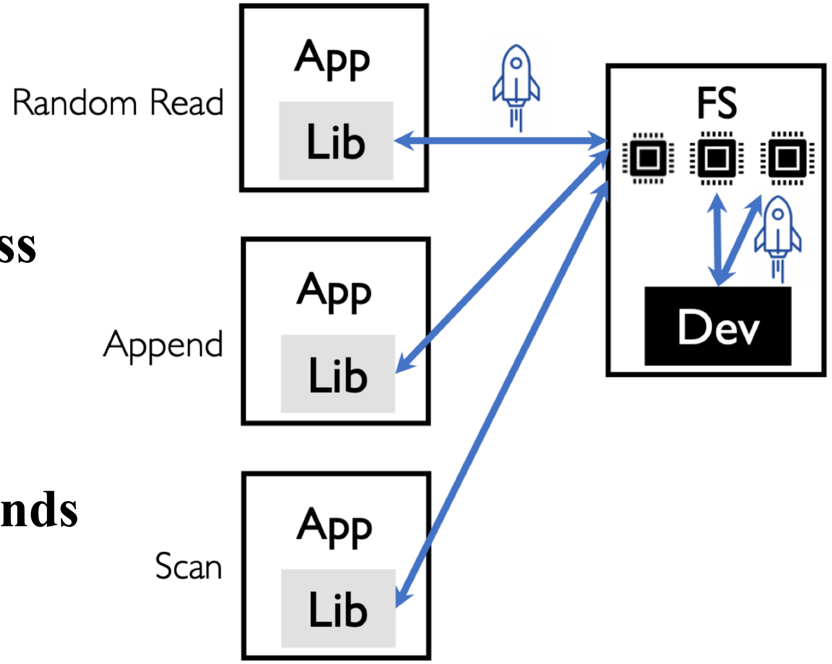
- Challenge
  - Base Performance
    - Inter-process communication & device access
  - Scale up and down
    - Scalability
    - Dynamic and heterogeneous application demands
    - Invest just-right amount of CPU



Background

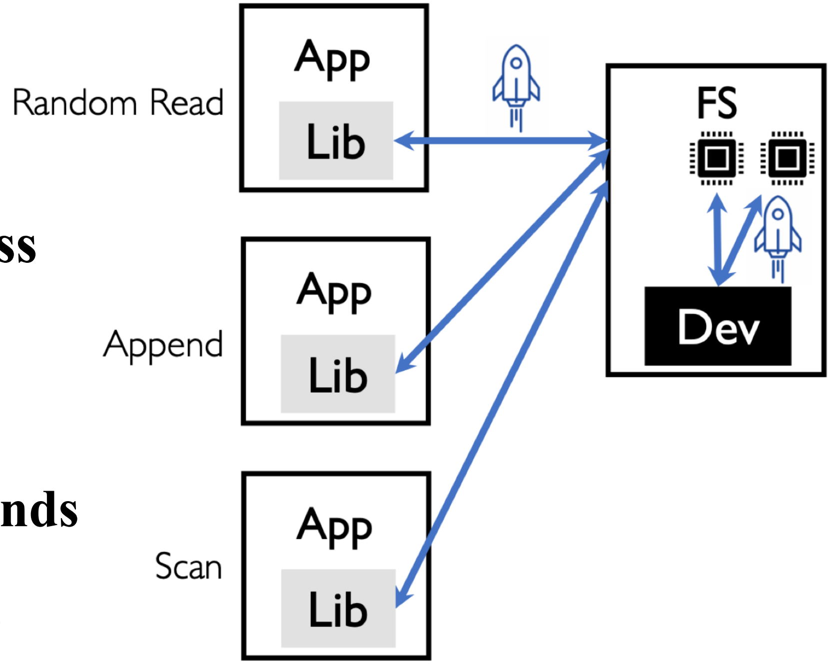
USTC, CHINA ADSLAB

- Challenge
  - Base Performance
    - Inter-process communication & device access
  - Scale up and down
    - Scalability
    - Dynamic and heterogeneous application demands
    - Invest just-right amount of CPU



Background

- Challenge
  - Base Performance
    - Inter-process communication & device access
  - Scale up and down
    - Scalability
    - Dynamic and heterogeneous application demands
    - Invest just-right amount of CPU



Outline

USTC, CHINA ADSLAB

- Background
- **uFS Design**
- Evaluation
- Concusion

uFS Design

USTC, CHINA ADSLAB

- **Single-Threaded uServer**
- **Multi-Threaded uServer**
- **Dynamic Load Management**
- **Employ Non-blocking Shared Structures Judiciously**

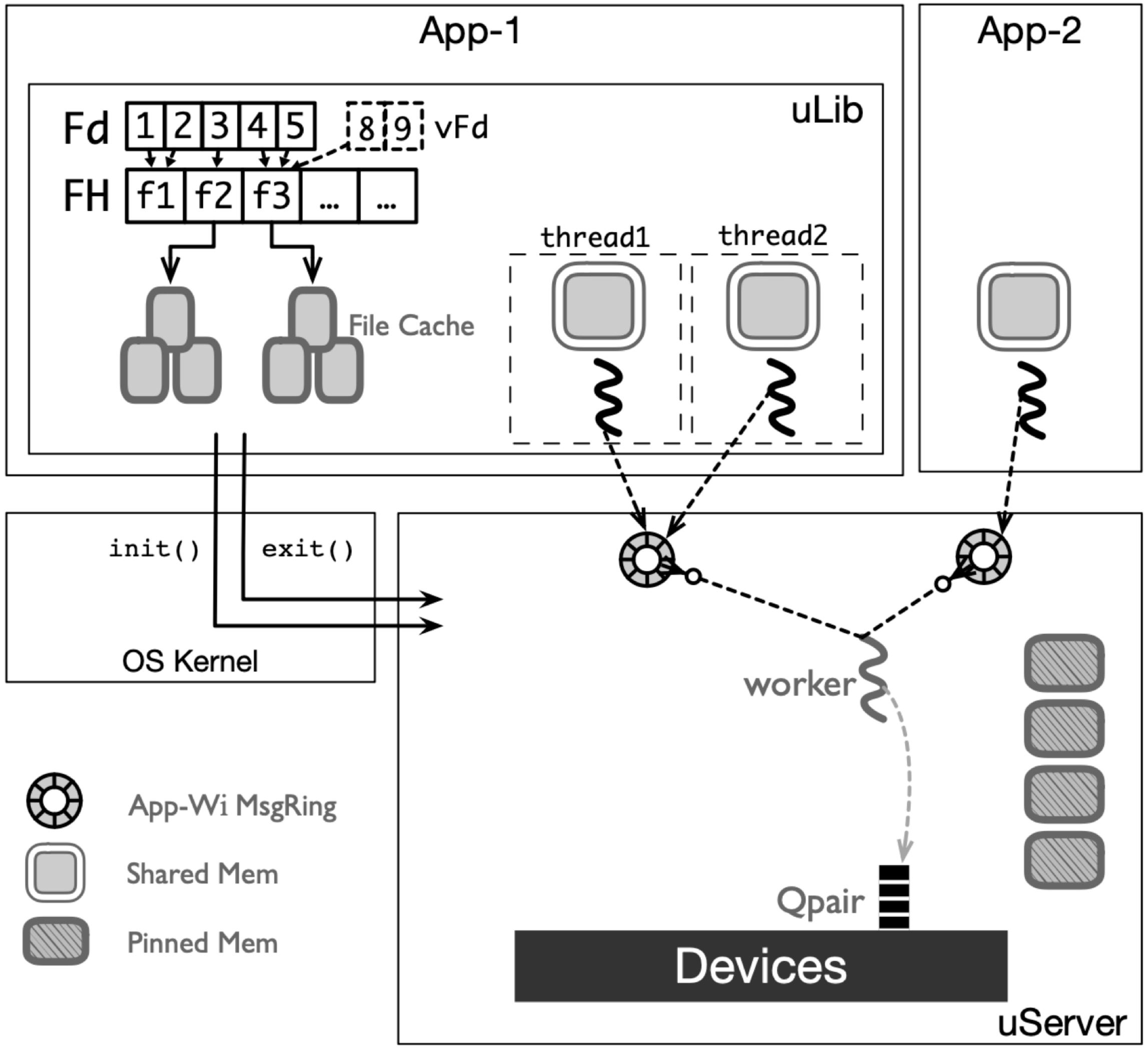
uFS Design

USTC, CHINA ADSLAB

- **Single-Threaded uServer**
- **Multi-Threaded uServer**
- **Dynamic Load Management**
- **Employ Non-blocking Shared Structures Judiciously**

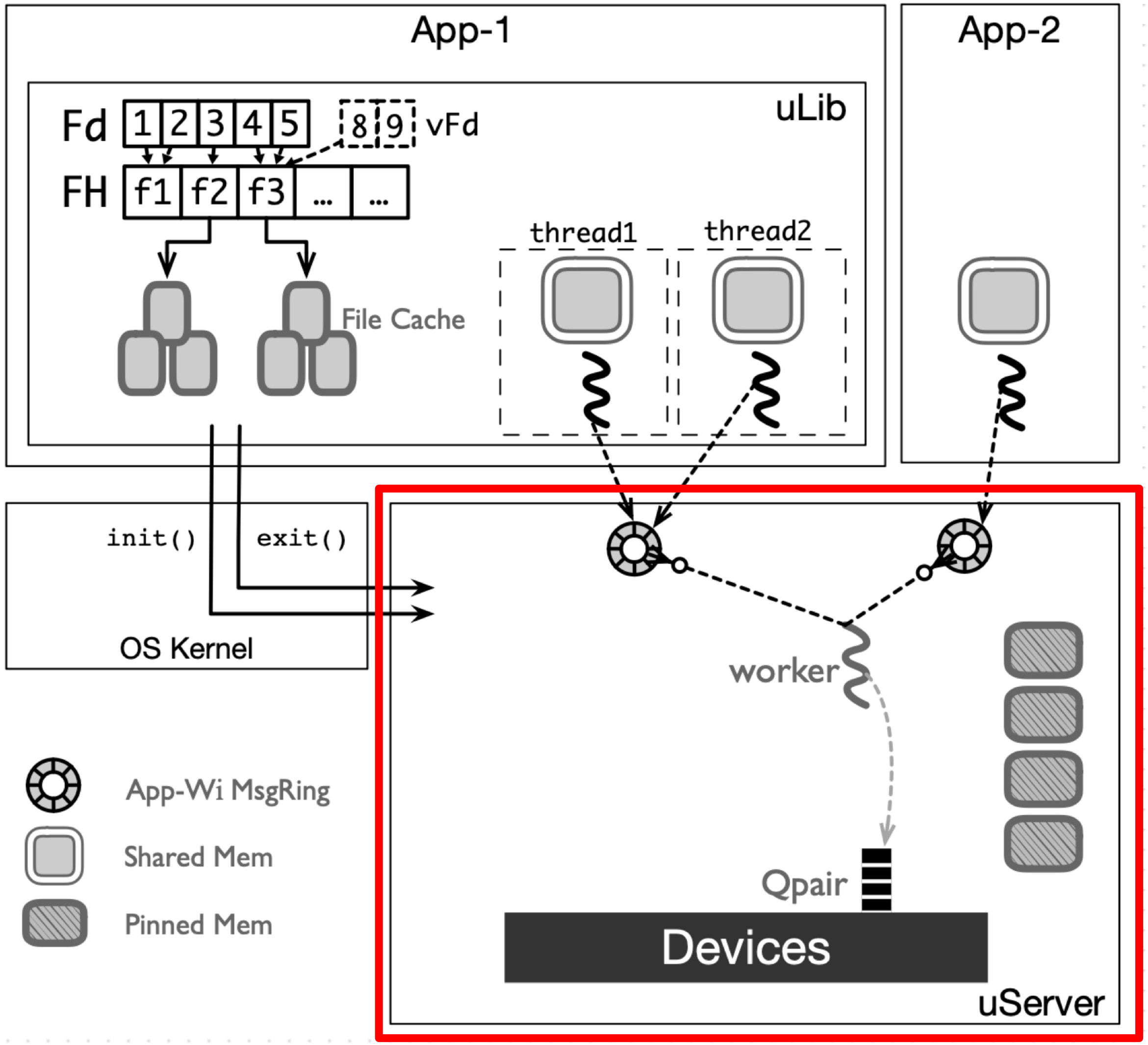
uFS Design

- **Single-Threaded uServer**



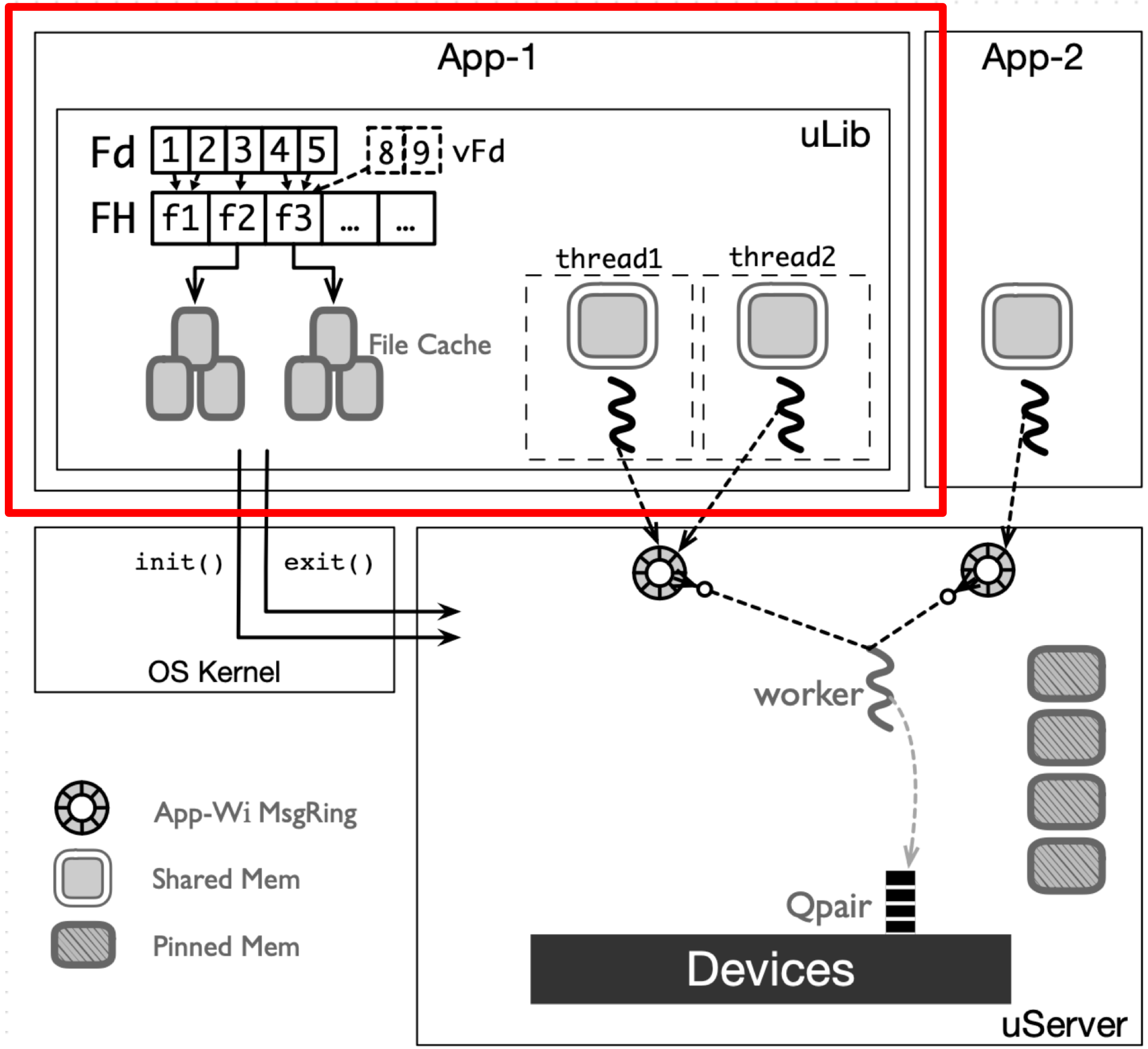
uFS Design

- **Single-Threaded uServer**
  - **uServer**
    - **Directly accesses the device via SPDK**
    - **Non-blocking polling**
    - **Pinned memory as block buffer cache**



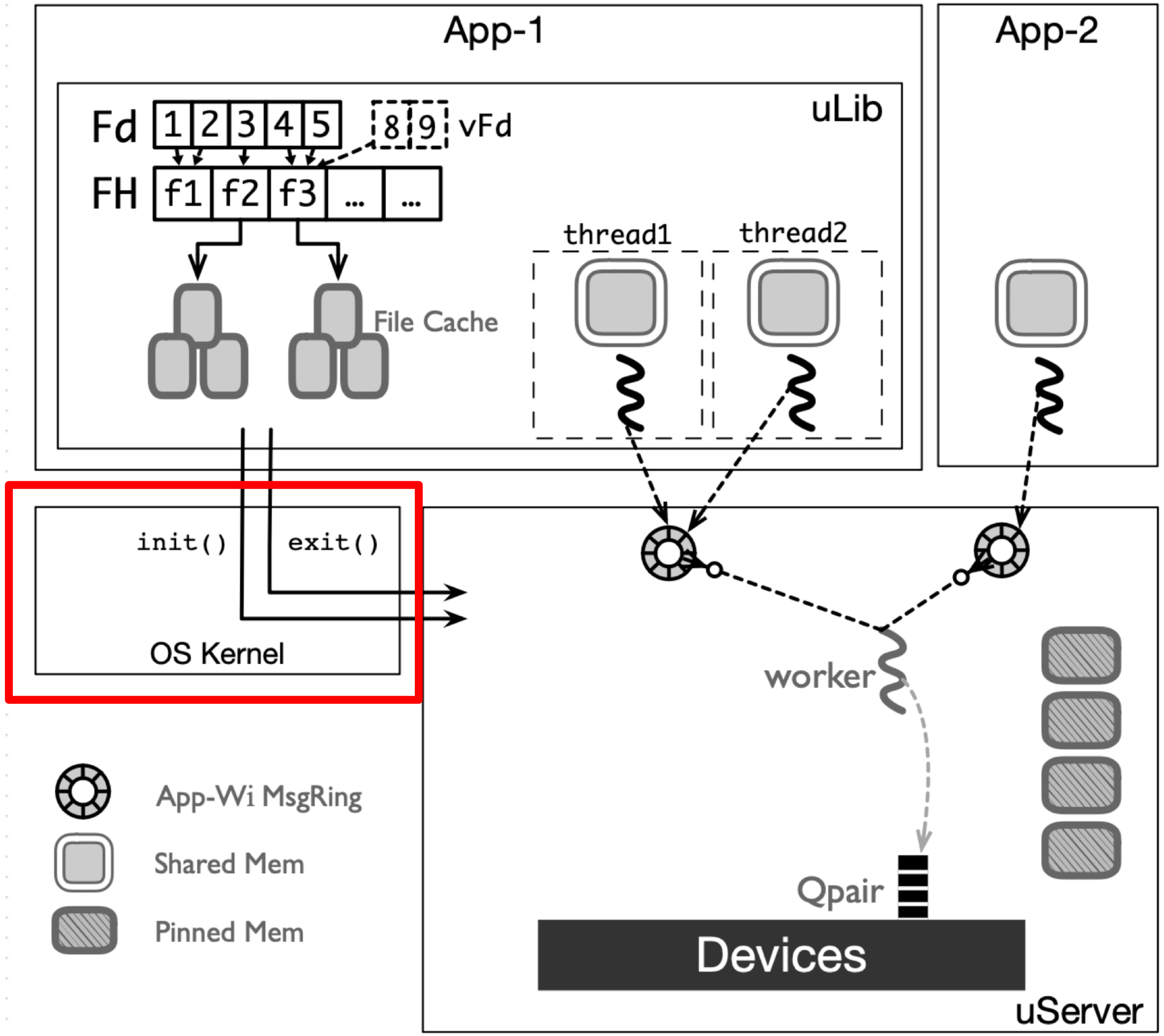
uFS Design

- Single-Threaded uServer
  - uLib
    - POSIX-API
    - App-integrated file cache (lease-based)
    - Open-lease management



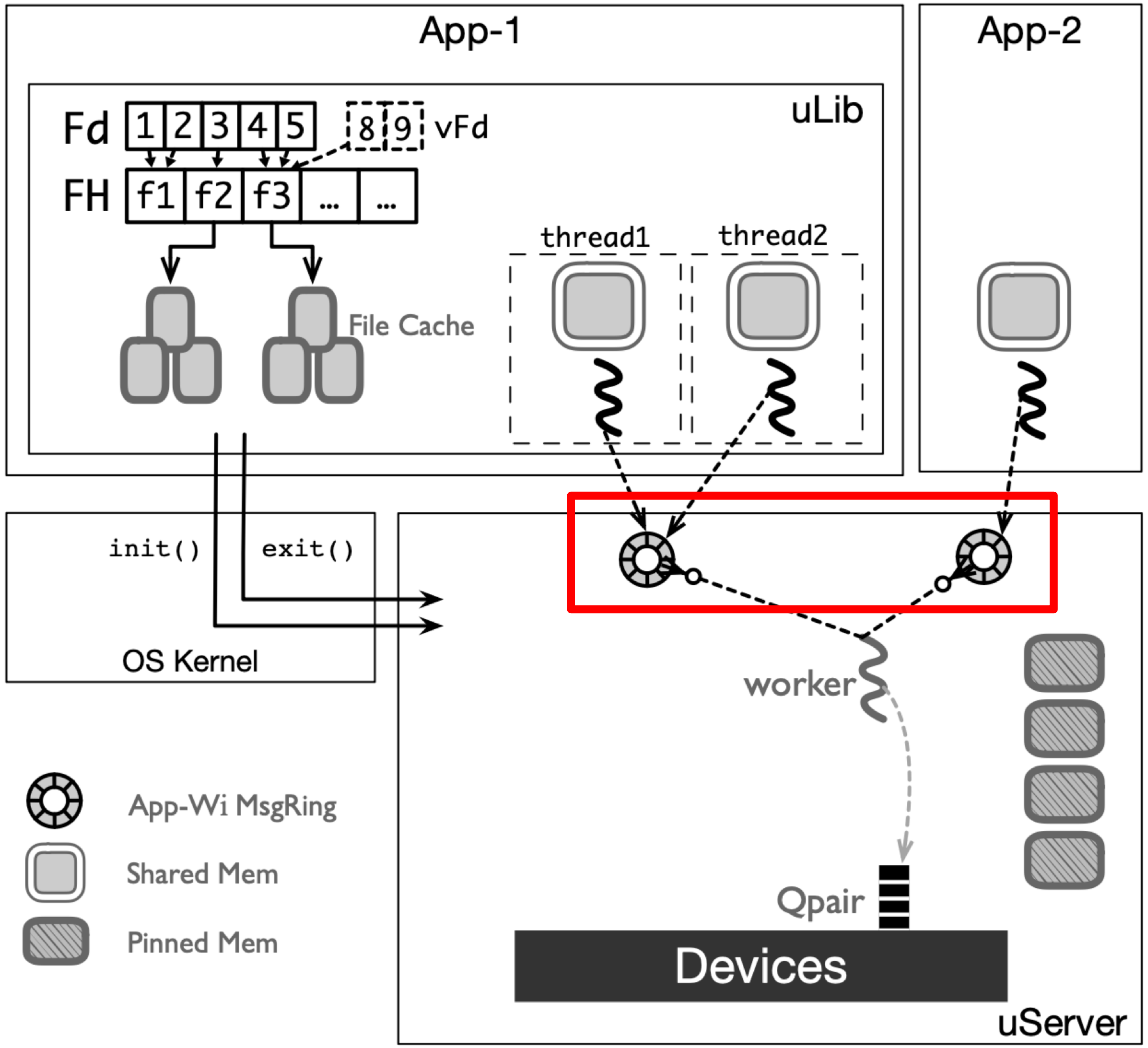
uFS Design

- **Single-Threaded uServer**
  - **The OS kernel only involves for initial authentication (fs_init)**



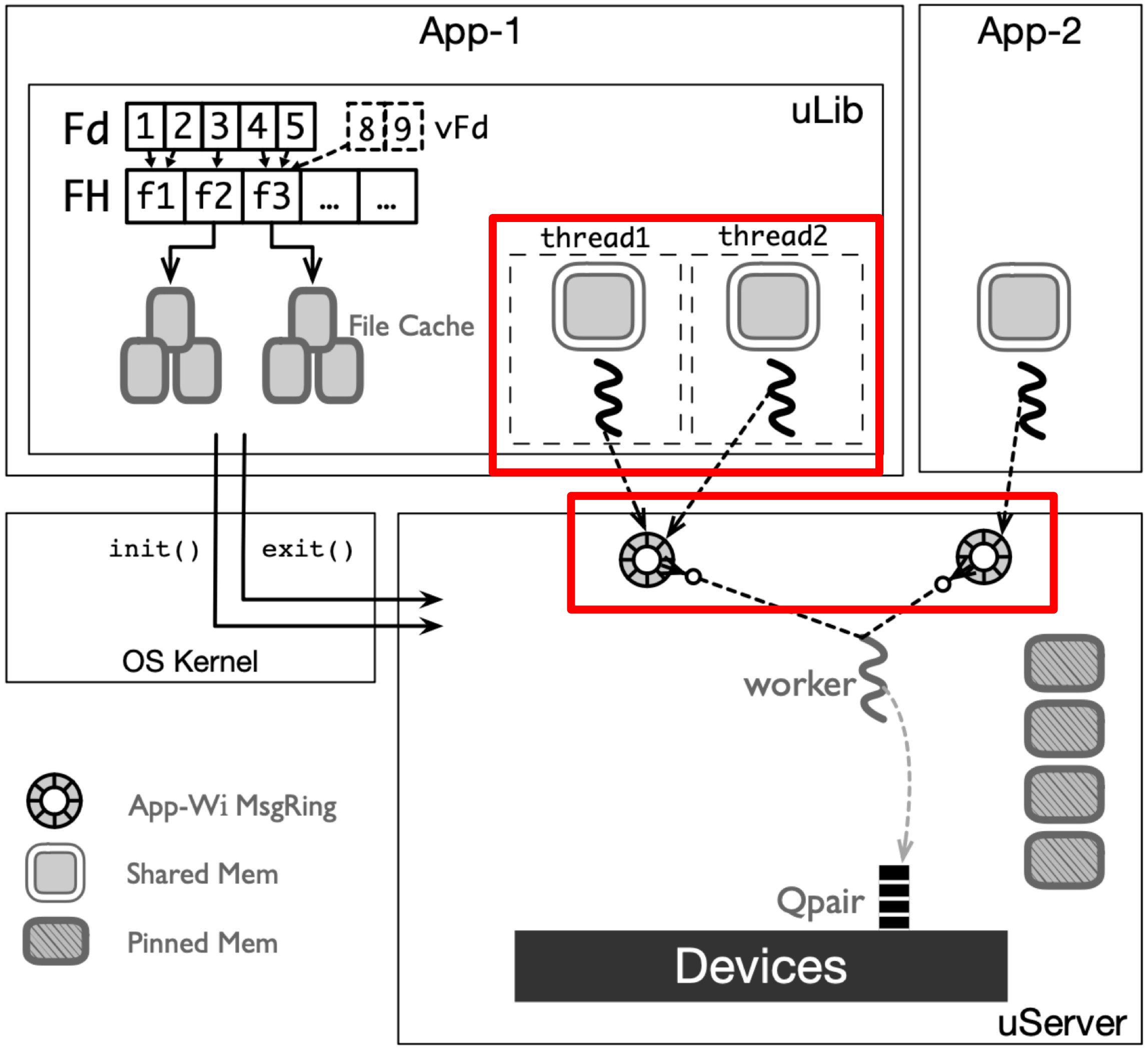
uFS Design

- Single-Threaded uServer
  - Inter-process communication
    - Control: shared-mem IPC
      - Cache-line-size message



uFS Design

- Single-Threaded uServer
  - Inter-process communication
    - Control: shared-mem IPC
      - Cache-line-size message
    - Data: customized malloc in uLib
      - uLib shares pages with uServer

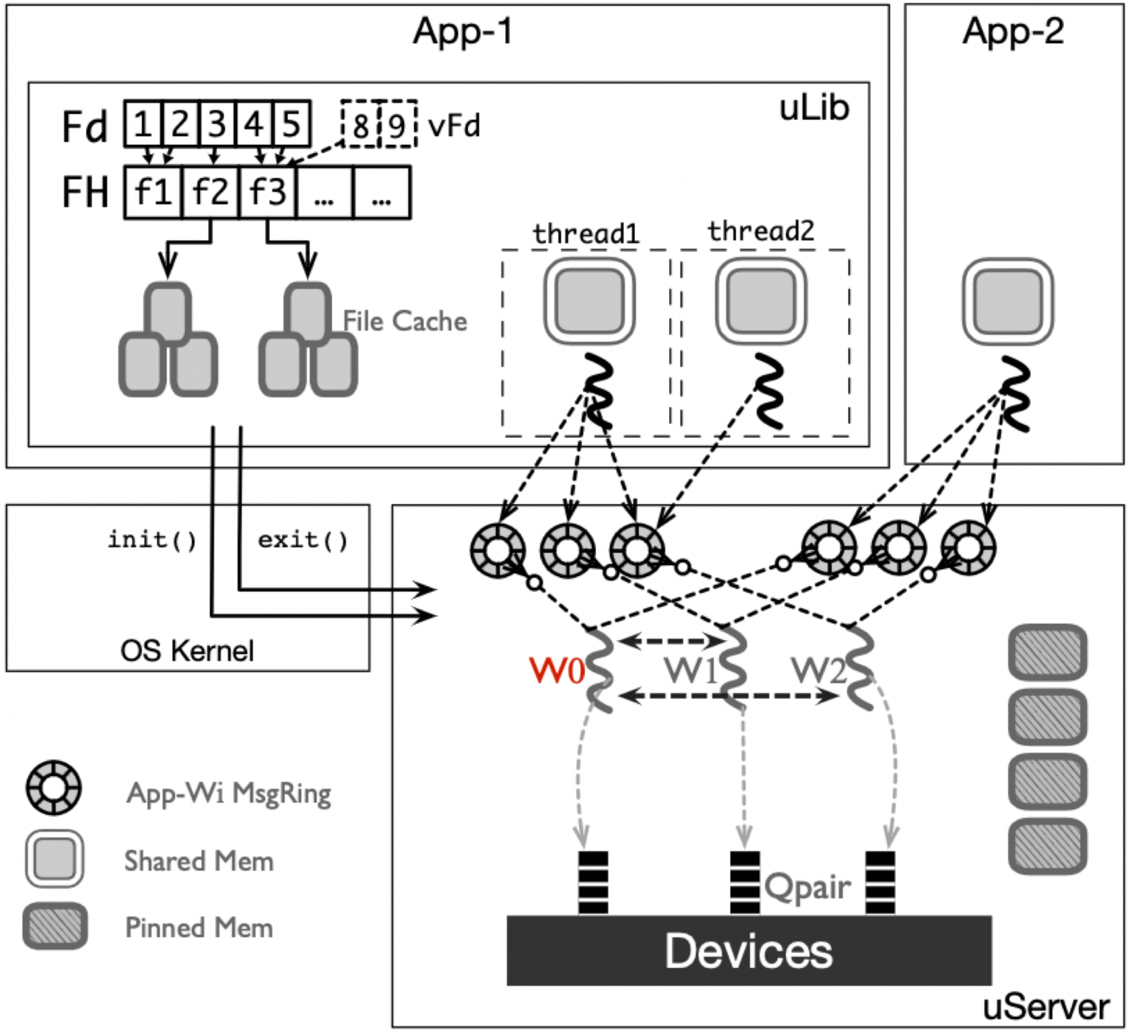


uFS Design

- Single-Threaded uServer
- **Multi-Threaded uServer**
- Dynamic Load Management
- Employ Non-blocking Shared Structures Judiciously

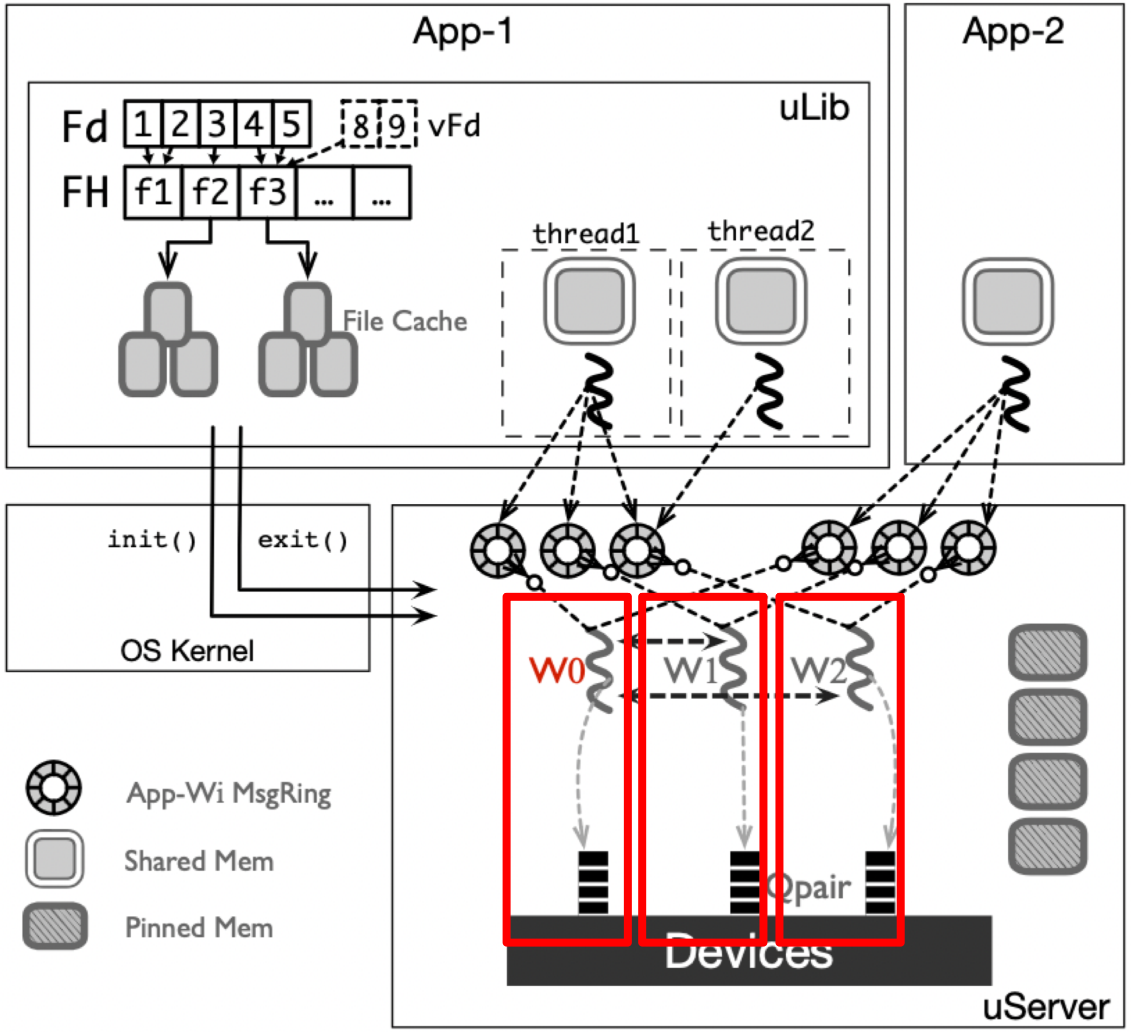
uFS Design

- **Multi-Threaded uServer**
  - **Utilize the full bandwidth of current I/O devices**
  - **More computation resource**



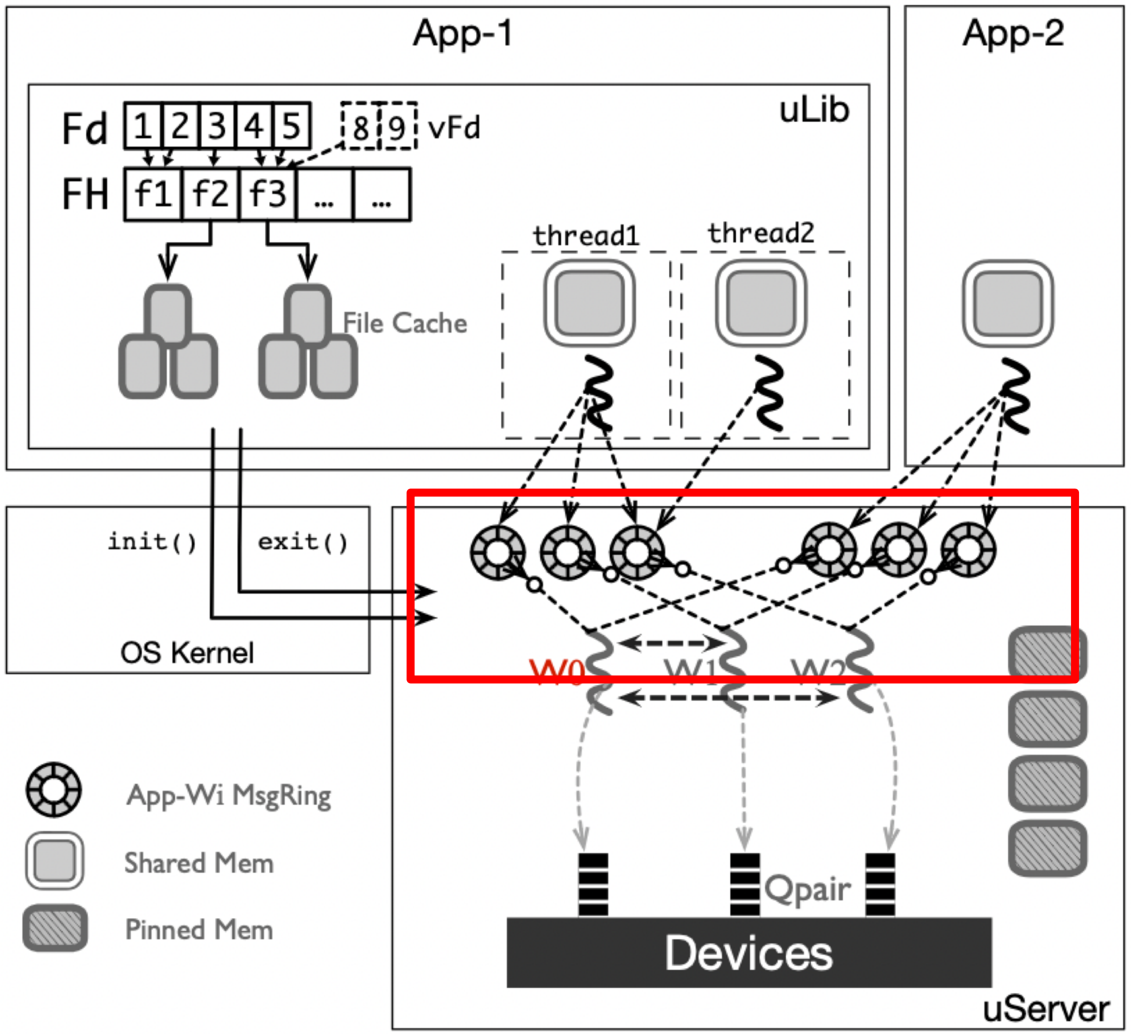
uFS Design

- **Multi-Threaded uServer**
  - **Scalable by design: sharing nothing**
  - **Each worker has several private data structure**
    - **Device requests qpair**



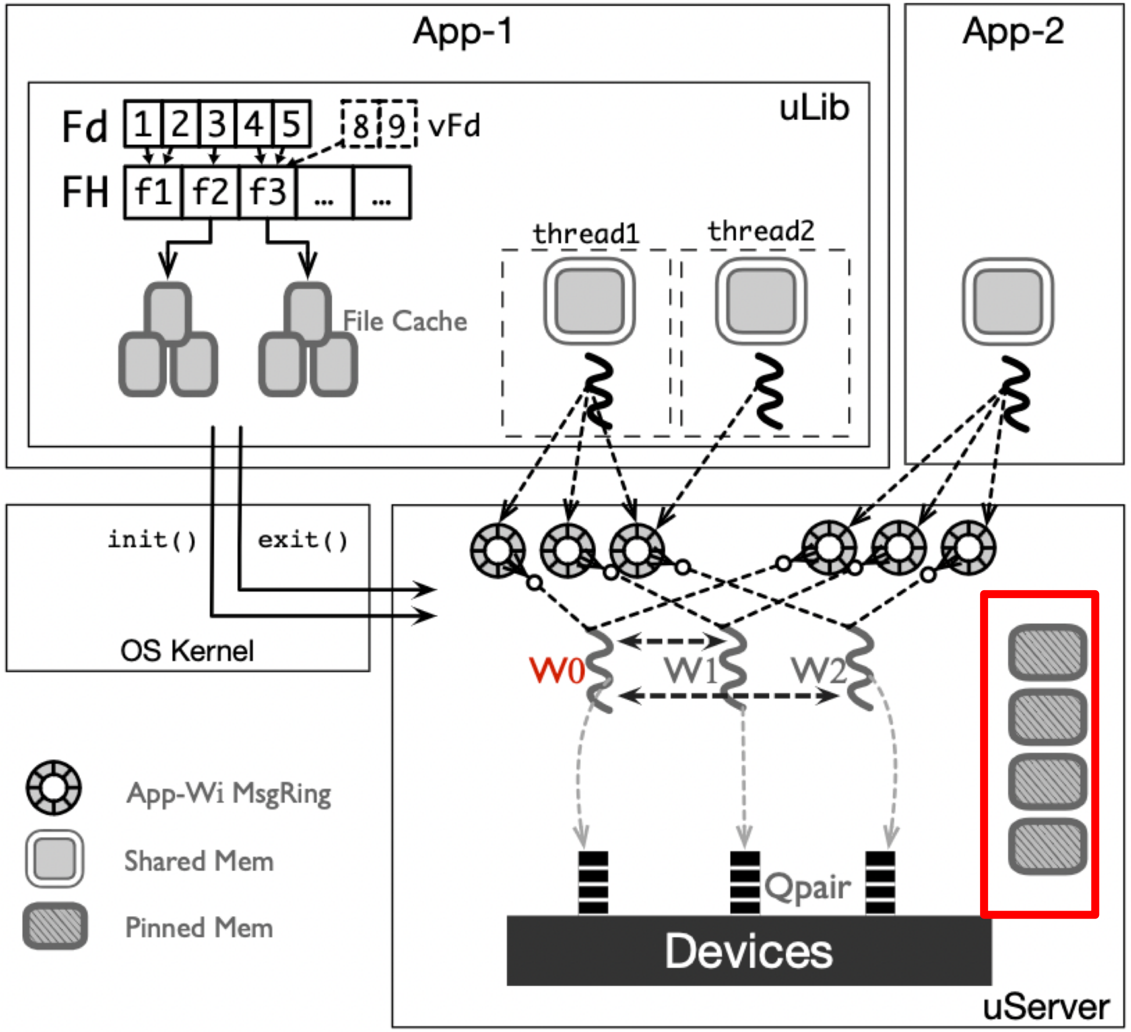
uFS Design

- **Multi-Threaded uServer**
  - **Scalable by design: sharing nothing**
  - **Each worker has several private data structure**
    - **Device requests qpair**
    - **Msg rings buffer per apps**



uFS Design

- **Multi-Threaded uServer**
  - **Scalable by design: sharing nothing**
  - **Each worker has several private data structure**
    - **Device requests qpair**
    - **Msg rings buffer per apps**
    - **Block buffer cache**



uFS Design

USTC, CHINA ADSLAB

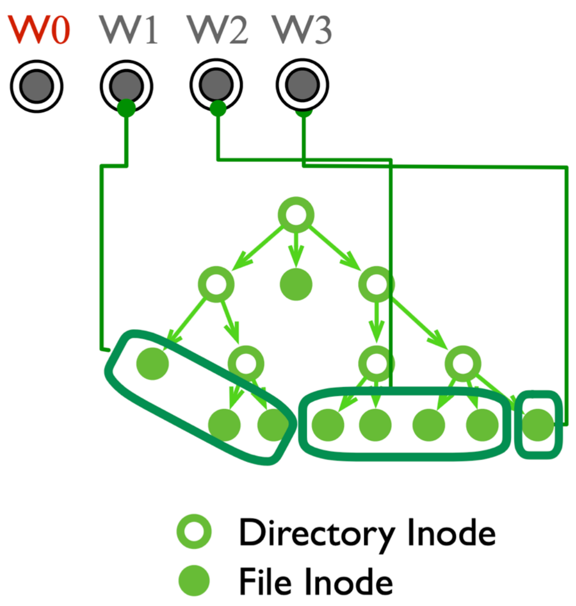
- **Multi-Threaded uServer**
  - **Data parallelism for scalability**
    - **Shared-nothing architecture**
    - **Divide filesystem states and data into threads**
    - **minimizes the sharing of in-memory data structures across cores**

Runtime Inode Ownership

Employ Non-blocking Shared Structures Judiciously

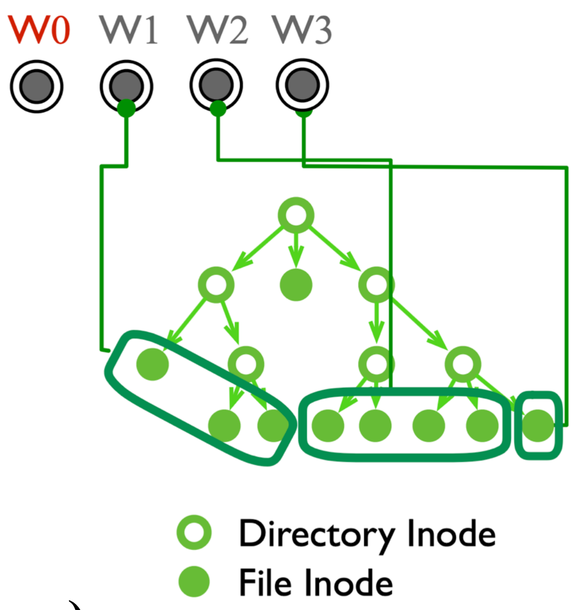
uFS Design

- Multi-Threaded uServer
  - Runtime Inode Ownership
    - Each group of inodes is exclusively accessed by one worker
      - No need for synchronization
      - Pre-assign data bitmap to each worker for data allocation



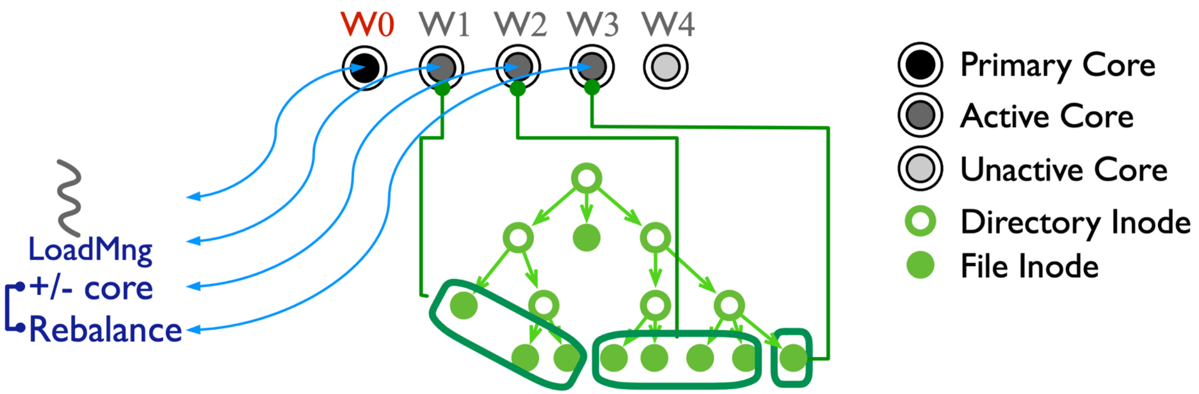
uFS Design

- **Multi-Threaded uServer**
  - **Runtime Inode Ownership**
    - **Each group of inodes is exclusively accessed by one worker**
      - **No need for synchronization**
      - **Pre-assign data bitmap to each worker for data allocation**
    - **Asymmetric Workers**
      - **Primary(W0)**
        - **Own and handles metadata workload (directory operations)**
        - **Coordinates with the workers**
      - **Worker**
        - **File operations**



Dynamic Load Management

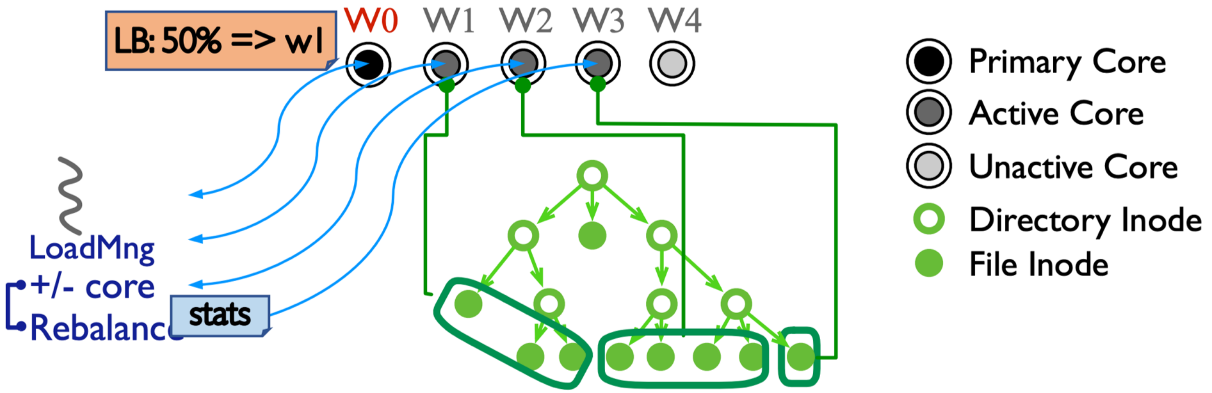
- Separate load managing thread (LoadMng)
  - Periodically gathers load stats from each worker (a monitoring window)
  - Decides per-worker [load goal] → Informs each worker the desired goal
  - Decides number of cores → Activate/Deactivate cores
- Worker invokes inode reassignment
  - Tracks per-inode stats
  - Given [load goal], decides which groups of inodes to be re-assigned



Dynamic Load Management

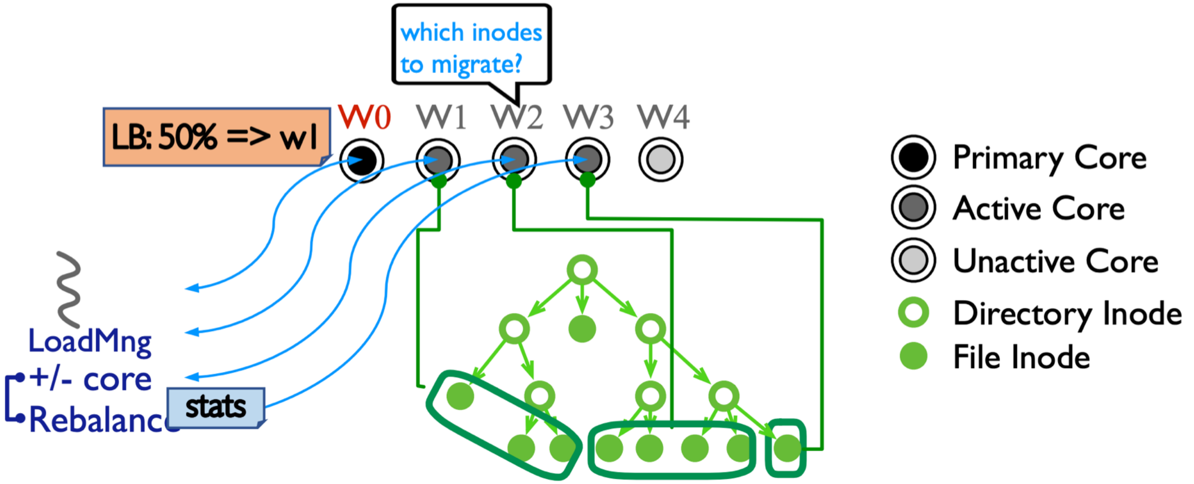
USTC, CHINA ADSLAB

- Separate load managing thread (LoadMng)
  - Periodically gathers load stats from each worker (a monitoring window)
  - Decides per-worker [load goal] → Informs each worker the desired goal
  - Decides number of cores → Activate/Deactivate cores
- Worker invokes inode reassignment
  - Tracks per-inode stats
  - Given [load goal], decides which groups of inodes to be re-assigned



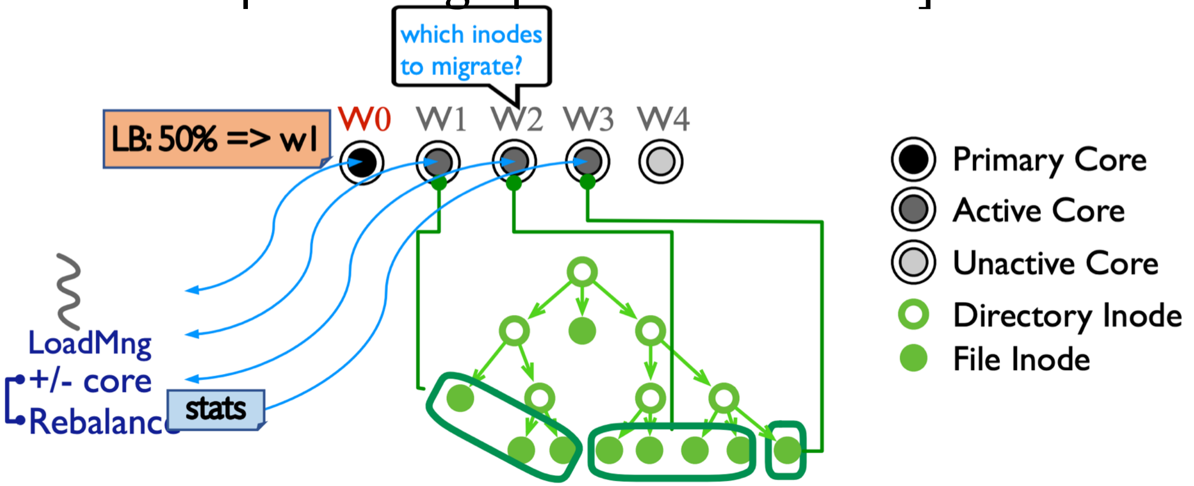
Dynamic Load Management

- Separate load managing thread (LoadMng)
  - Periodically gathers load stats from each worker (a monitoring window)
  - Decides per-worker [load goal] → Informs each worker the desired goal
  - Decides number of cores → Activate/Deactivate cores
- Worker invokes inode reassignment
  - Tracks per-inode stats
  - Given [load goal], decides which groups of inodes to be re-assigned



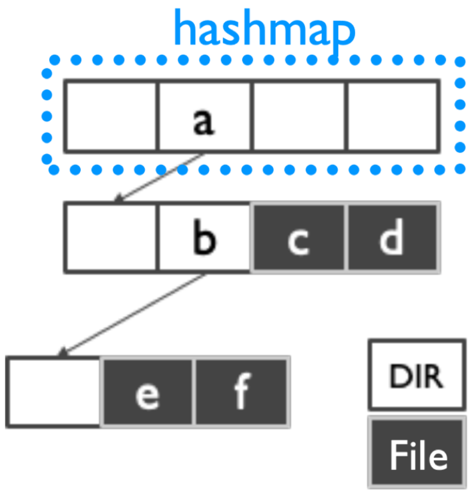
Dynamic Load Algorithms

- Load balancing
  - Towards minimizing congestion on each core
- Core allocation
  - Meets a per-core CPU utilization goal
  - Answer the "what if" questions by algorithmically emulating the load balancing results
    - Load balancing as a black-box
    - What if [add one core | no change | remove one core]



Employ Non-blocking Shared Structures Judiciously

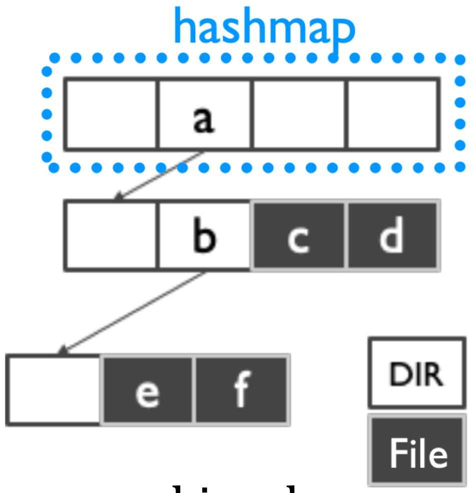
- Dentry Cache and Permission Checking
  - Recursive HashMap
  - Only the primary worker can update and all can read
  - Leverage industrial-quality lock-free data structures



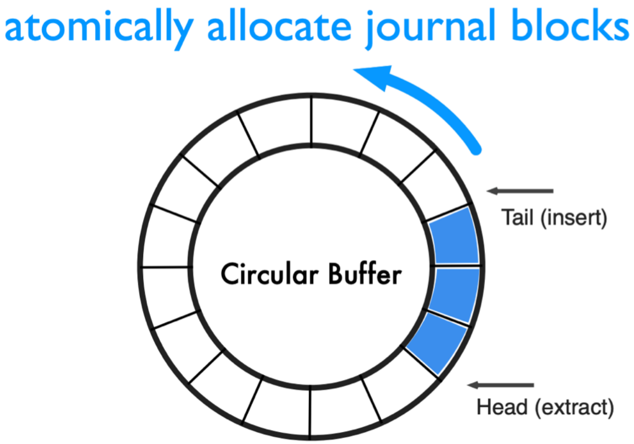
Employ Non-blocking Shared Structures Judiciously

USTC, CHINA ADSLAB

- Dentry Cache and Permission Checking
  - Recursive HashMap
  - Only the primary worker can update and all can read
  - Leverage industrial-quality lock-free data structures



- Global Logic Journal that allows maximal parallelism
  - Each worker can initialize journal transactions independently for owned inodes
  - Negligible overhead added
    - Recording logic modification is lightweight
    - Minimal critical section when reserving journal blocks



Outline

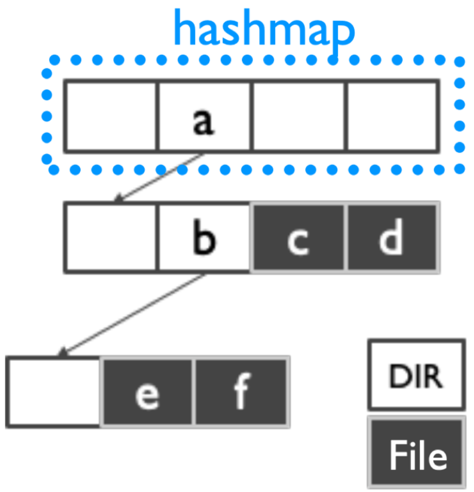
USTC, CHINA ADSLAB

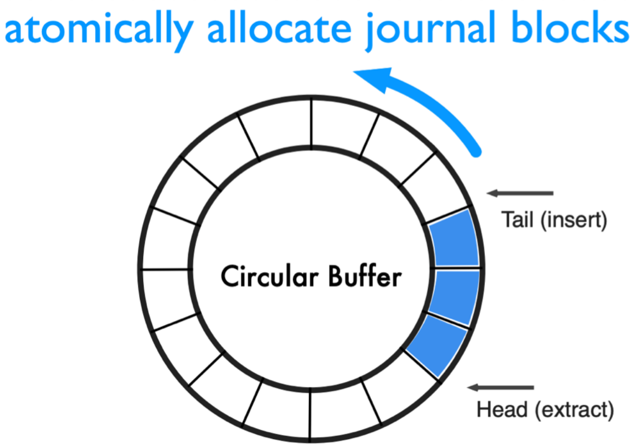
- Background
- uFS Design
- **Evaluation**
- Conclusion

Evaluation

- uFS offers good single-threaded base performance
- uFS performs well as a multi-threaded micro-kernel
- uFS dynamically scales to match demand
  - Load Balancing Experiments
  - Core Allocation Experiments
- uFS performs and scales well with real applications
  - LevelDB and YCSB workloads

- Platform
  - Intel Optane 905P SSD; Intel® Xeon® Gold 5218R CPU
  - Linux 5.4, SPDK 18.04

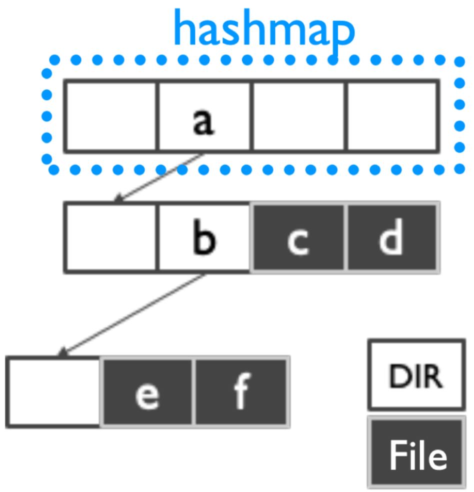


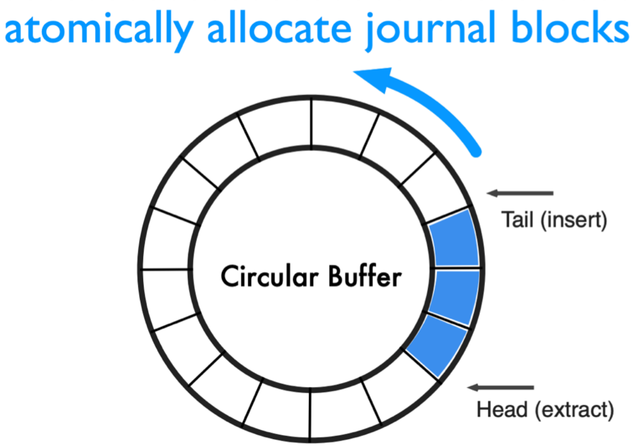


Evaluation

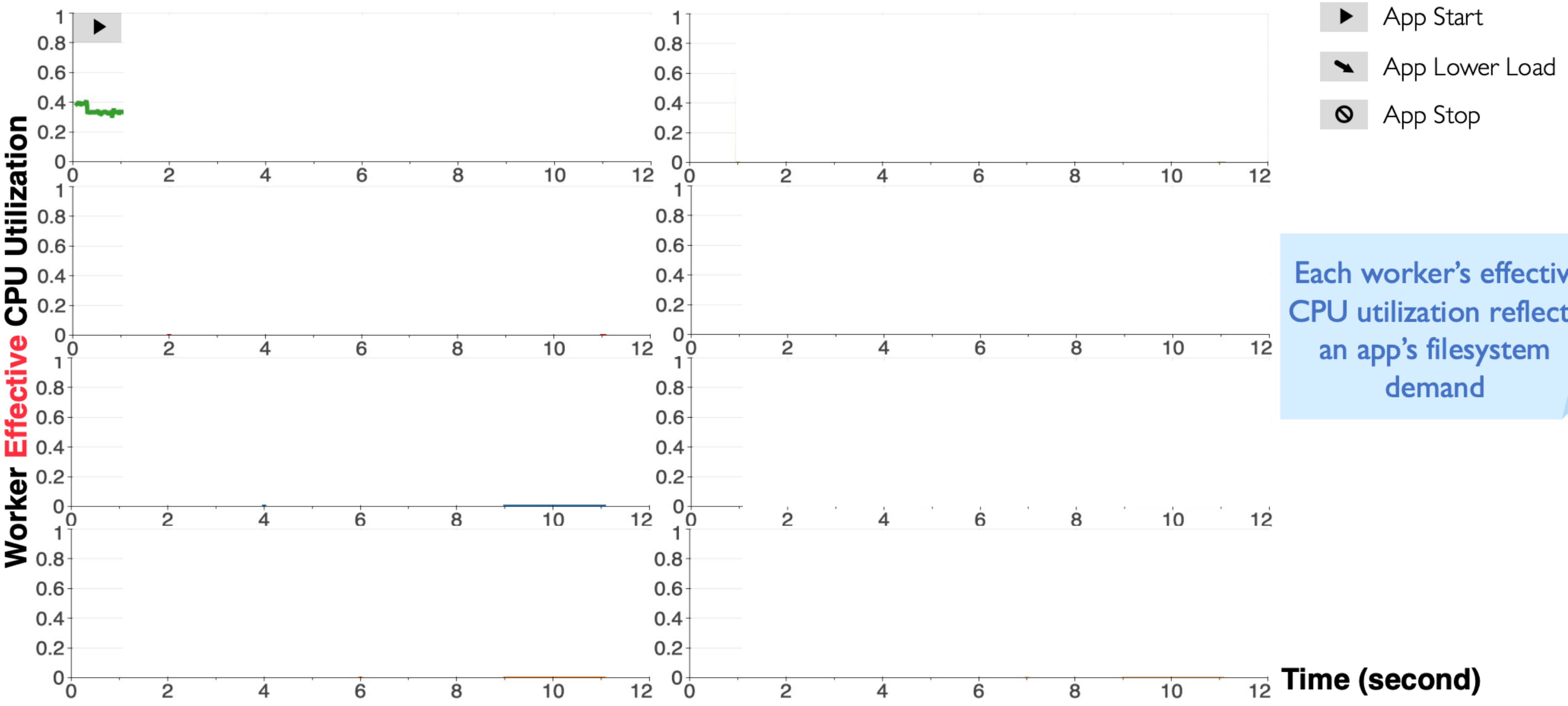
- uFS offers good single-threaded base performance
- uFS performs well as a multi-threaded micro-kernel
- uFS dynamically scales to match demand
  - Load Balancing Experiments
  - Core Allocation Experiments
- uFS performs and scales well with real applications
  - LevelDB and YCSB workloads

- Platform
  - Intel Optane 905P SSD; Intel® Xeon® Gold 5218R CPU
  - Linux 5.4, SPDK 18.04





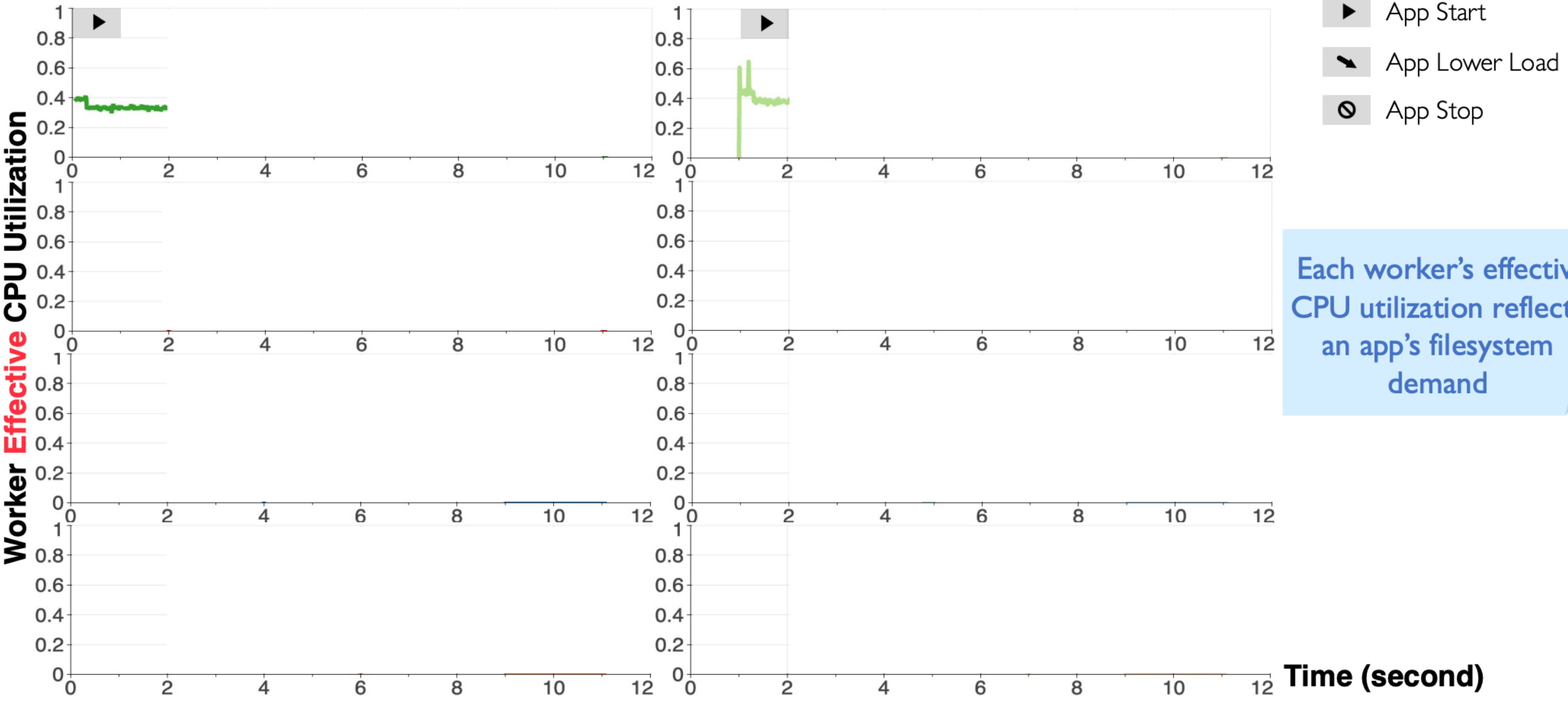
Core Allocation Experiments



Each worker's effective CPU utilization reflects an app's filesystem demand

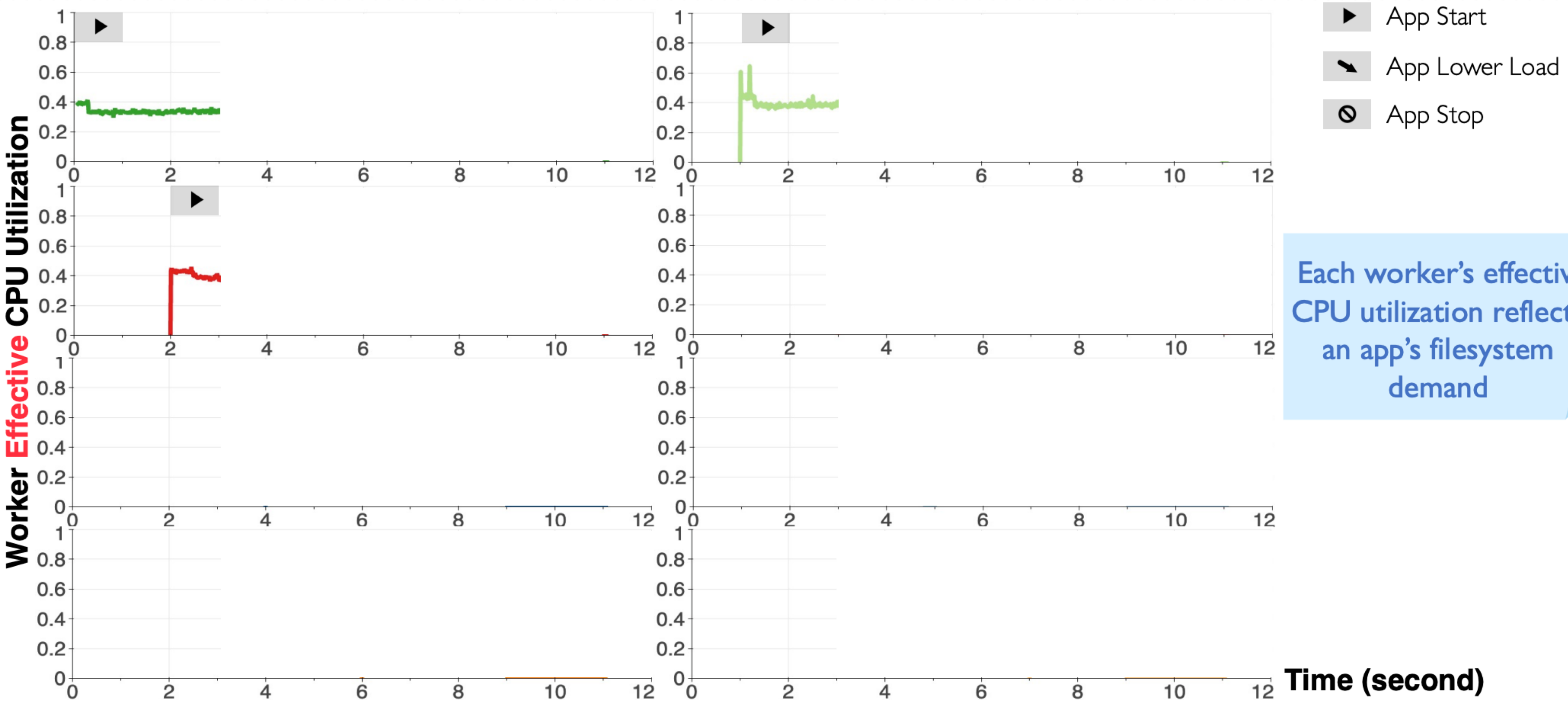
Core Allocation Experiments

USTC, CHINA ADSLAB



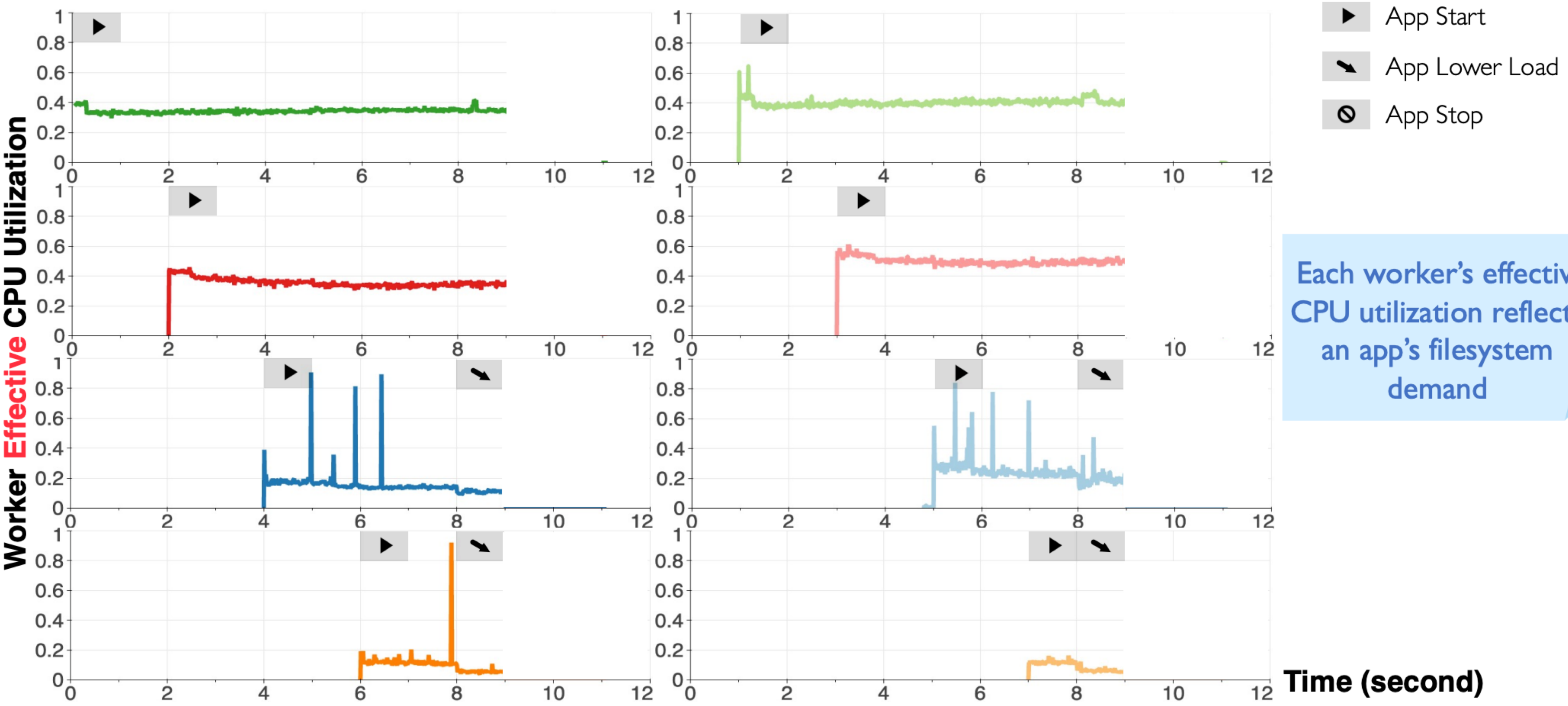
Each worker's effective CPU utilization reflects an app's filesystem demand

Core Allocation Experiments



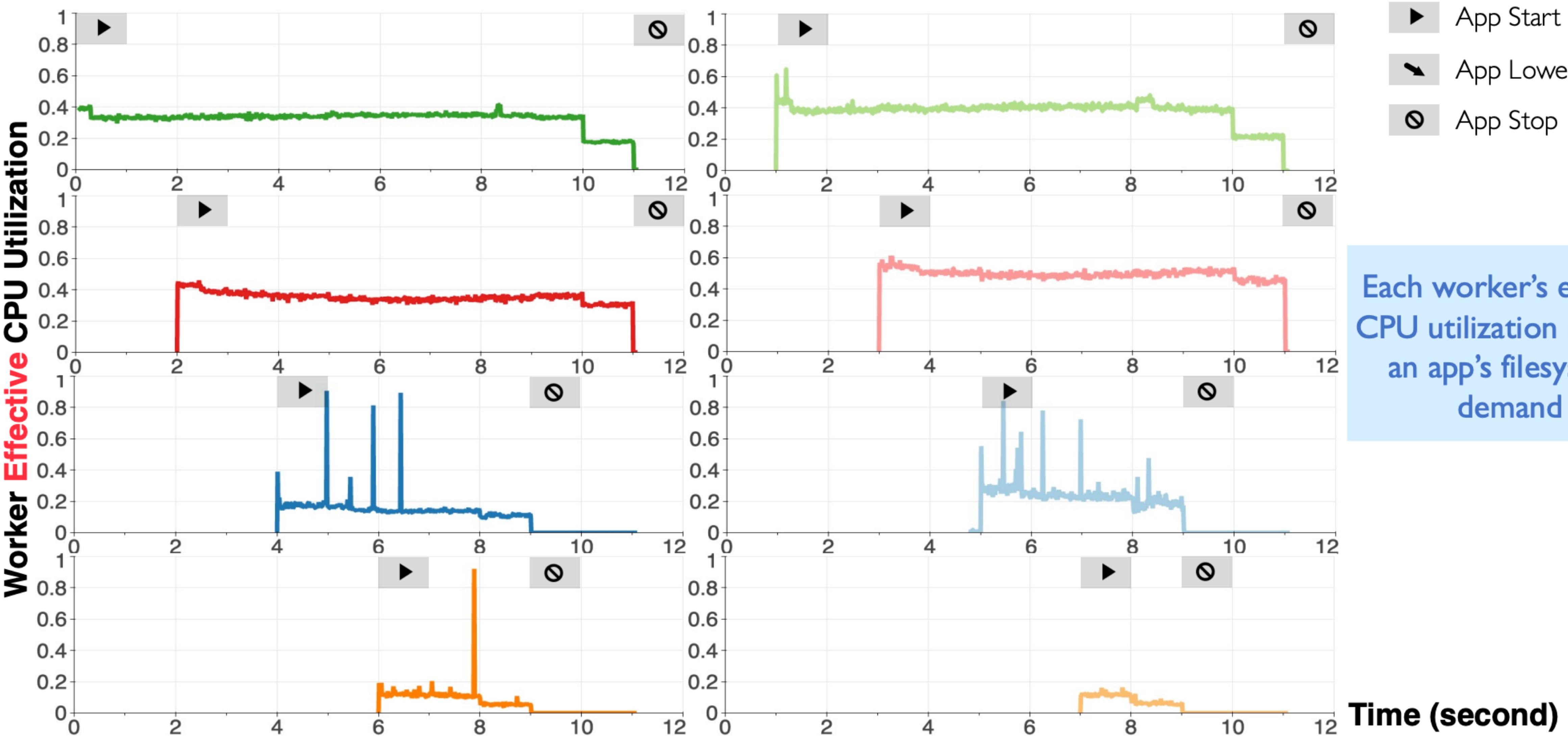
Each worker's effective CPU utilization reflects an app's filesystem demand

Core Allocation Experiments



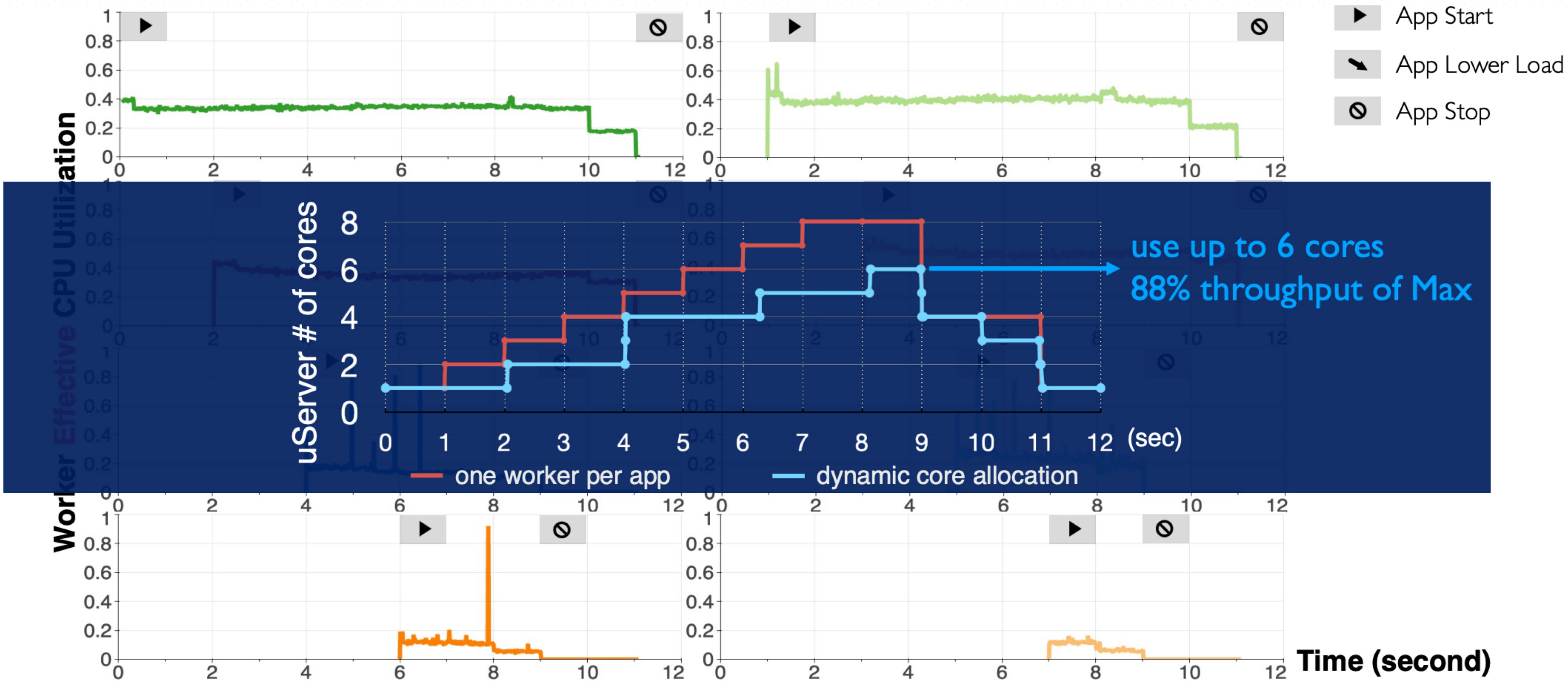
Each worker's effective CPU utilization reflects an app's filesystem demand

Core Allocation Experiments



Each worker's effective CPU utilization reflects an app's filesystem demand

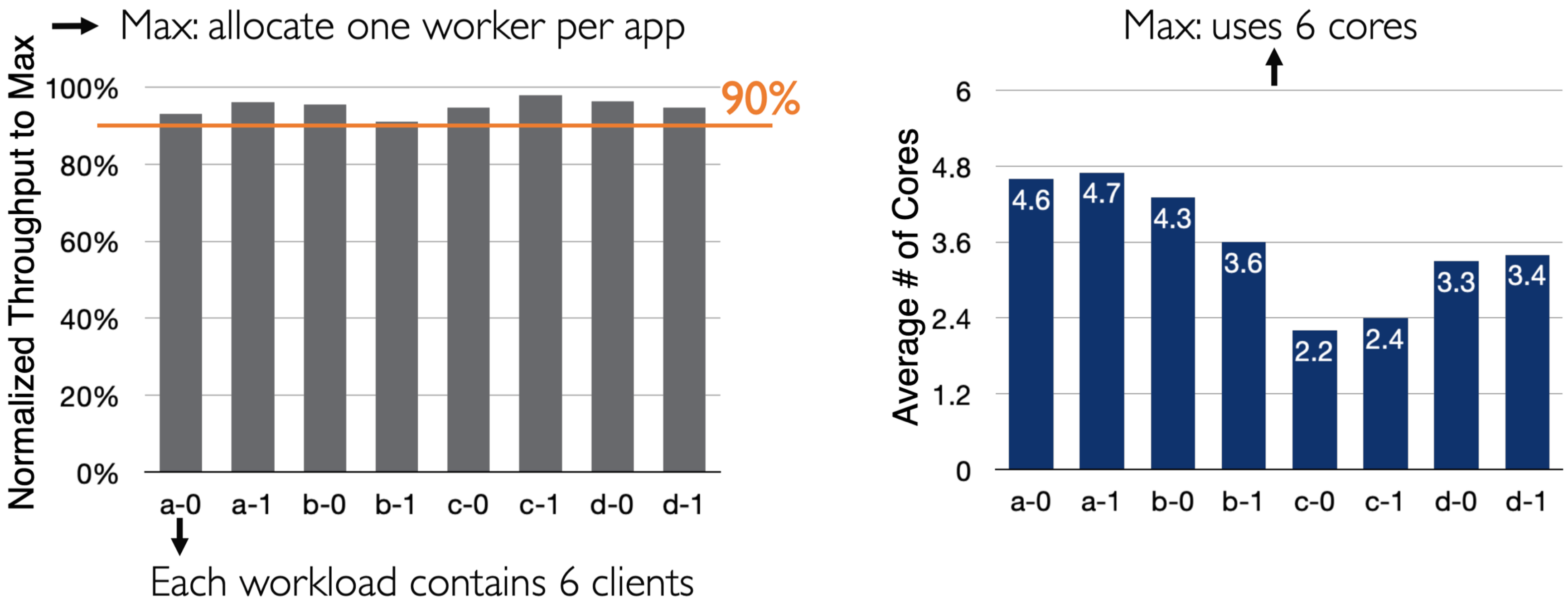
Core Allocation Experiments



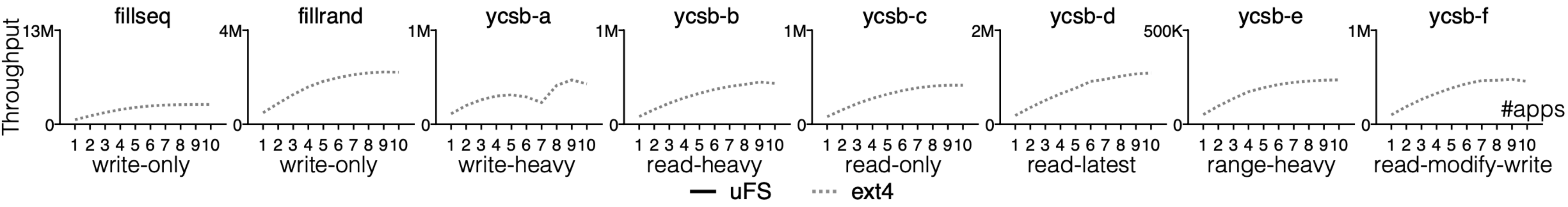
Core Allocation Experiments

8 workloads: each changes one factor by N steps along the time

- Factor example: think-time, data screw degree, request size
- uFS delivers between 91% to 98% throughput of Max
- uFS controls number of cores as needed

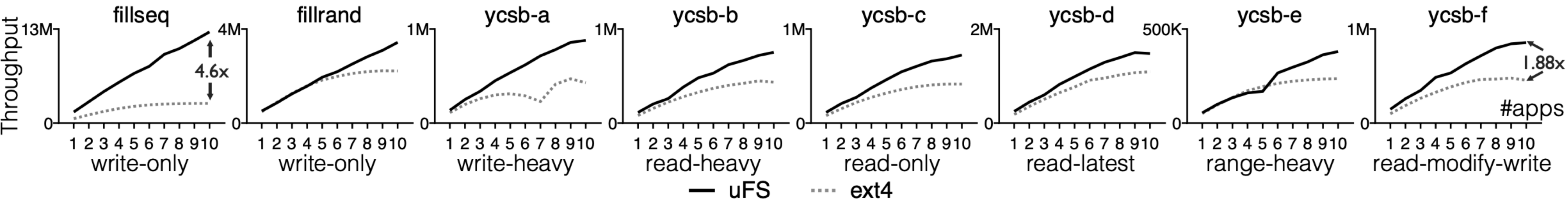


LevelDB: uFS with Real Apps



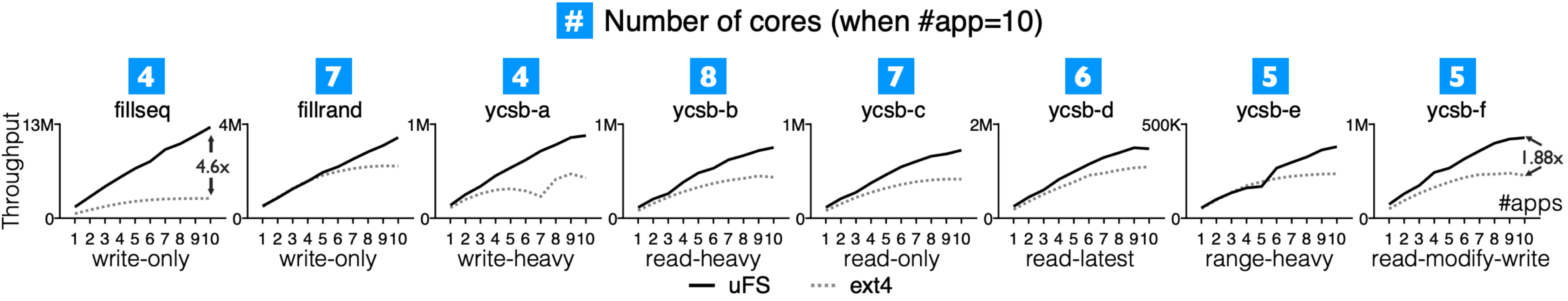
- uFS can scale much better than ext4
- uFS will allocate different number of cores for various workloads
- Giving more cores (>10) to ext4 does not help much for performance

LevelDB: uFS with Real Apps



- uFS can scale much better than ext4
- uFS will allocate different number of cores for various workloads
- Giving more cores (>10) to ext4 does not help much for performance

LevelDB: uFS with Real Apps



- uFS can scale much better than ext4
- uFS will allocate different number of cores for various workloads
- Giving more cores (>10) to ext4 does not help much for performance

Outline

- Background
- uFS Design
- Evaluation
- **Conclusion**

Conclusion

- **uFS: a filesystem semi-microkernel**
  - Designs for modern storage device performance delivery and scalability
    - Outperforms ext4 under LevelDB workloads by 1.22x to 4.6x
  - Scales independently from the applications and dynamically matches demand
- **Filesystem Semi-Microkernel Approach**
  - Performs and scales well under various workloads
  - Has all the benefits of user-level development

Conclusion

- **uFS: a filesystem semi-microkernel**
  - Designs for modern storage device performance delivery and scalability
    - Outperforms ext4 under LevelDB workloads by 1.22x to 4.6x
  - Scales independently from the applications and dynamically matches demand
- **Filesystem Semi-Microkernel Approach**
  - Performs and scales well under various workloads
  - Has all the benefits of user-level development