A Scheduling Framework that Makes any Disk Schedulers Non-work-conserving solely based on Request Characteristics

Yuehai Xu and **Song Jiang**

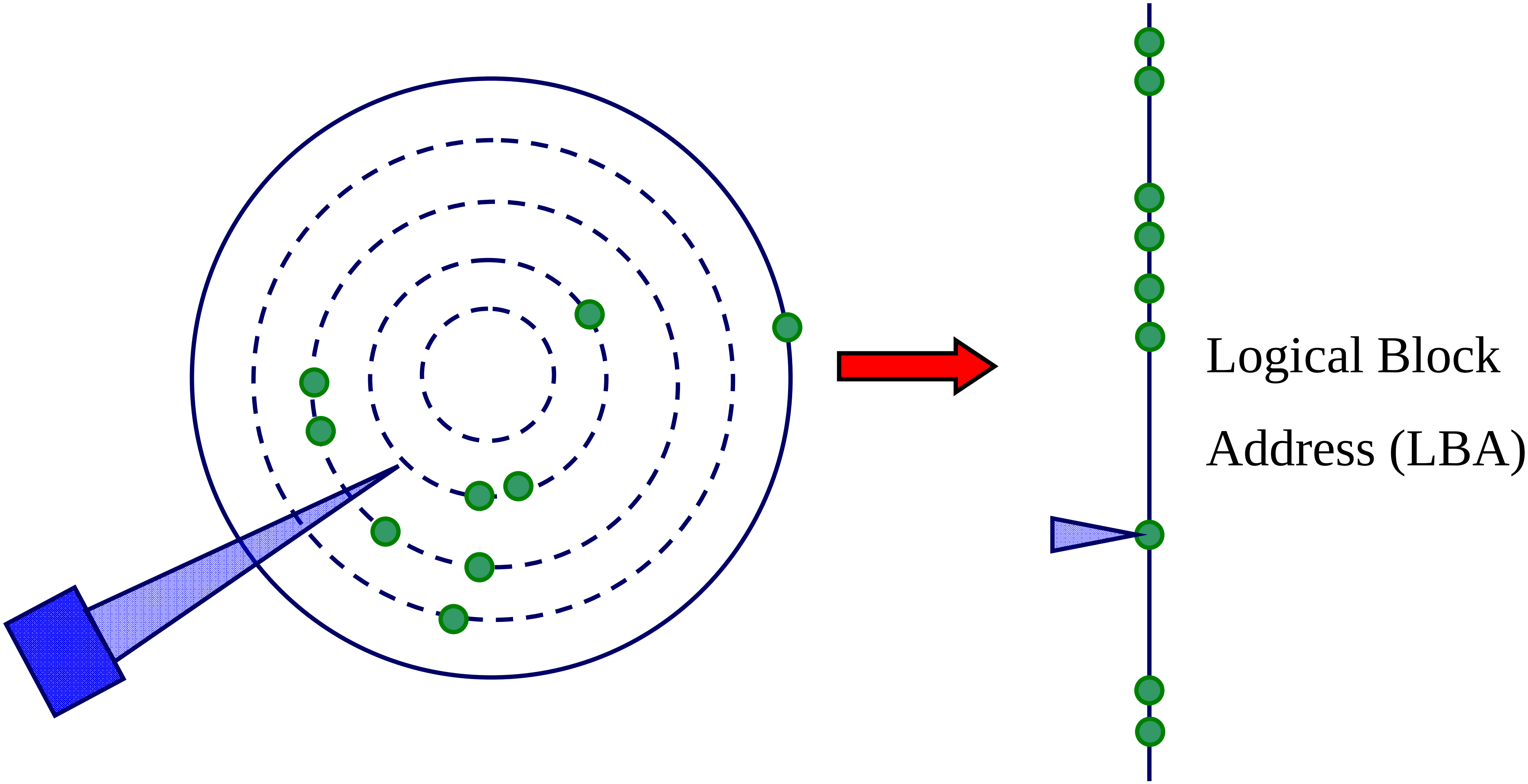
Department of Electrical and Computer Engineering

Wayne State University

Disk Performance and Workload Spatial Locality

- The disk is cost effective with its ever increasing capacity and peak throughput.
- The performance with non-sequential access is critical for the disk to be competitive.
  - Virtual machine environment
  - Consolidated storage system
- The effective performance depends on exploitation of spatial locality.
  - This locality is usually exploited statically in the request scheduling.
  - In this work, we exploit it in both space and time dimensions.

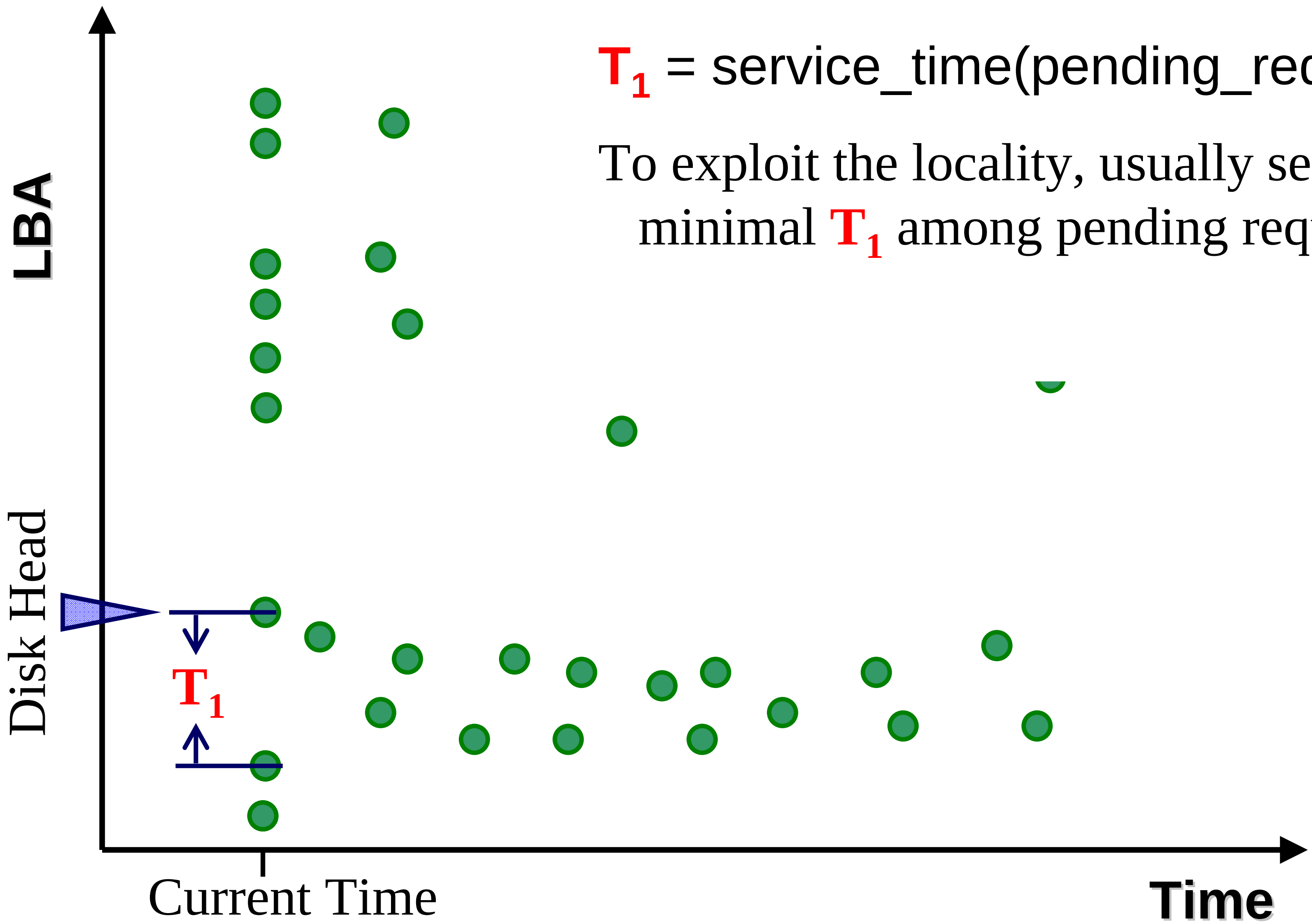
Quantifying Request Service Time



From 1-D Locality to 2-D Locality

T_1 = service_time(pending_request)

To exploit the locality, usually select minimal T_1 among pending requests.

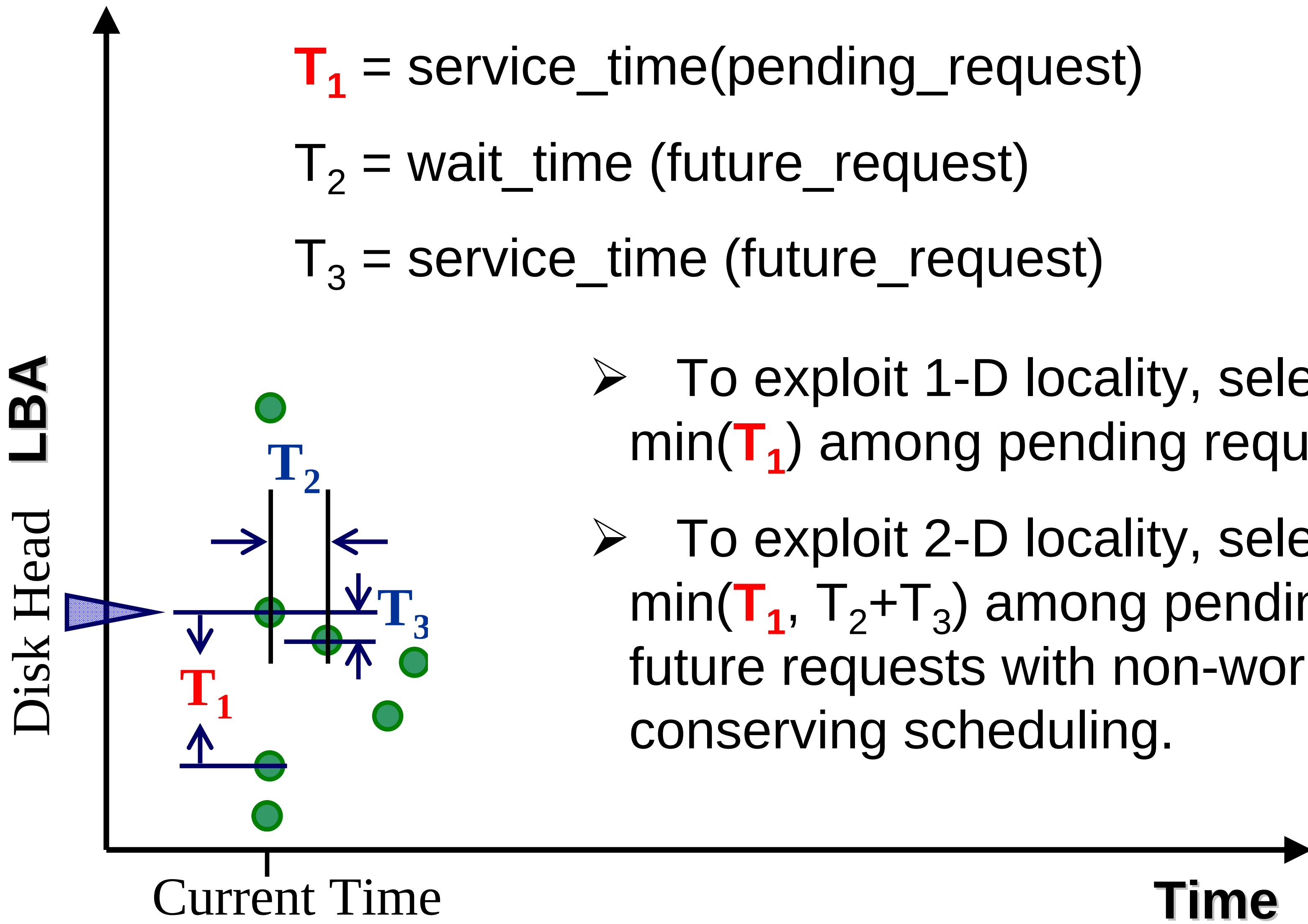


From 1-D Locality to 2-D Locality

T_1 = service_time(pending_request)

T_2 = wait_time (future_request)

T_3 = service_time (future_request)



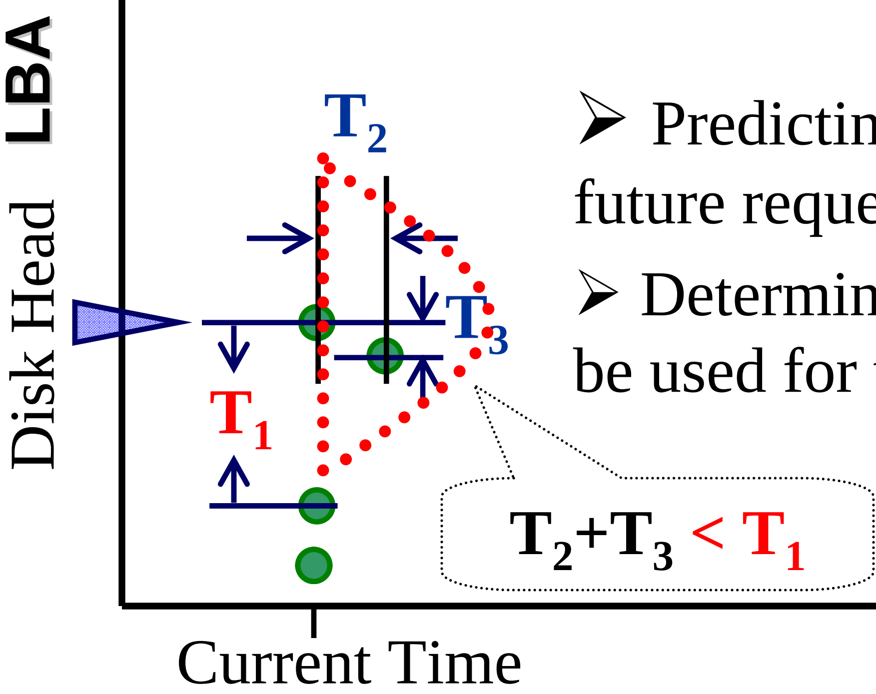
- To exploit 1-D locality, select min(T_1) among pending requests.
- To exploit 2-D locality, select min(T_1, T_2+T_3) among pending and future requests with non-work-conserving scheduling.

Challenges of Exploiting 2-D Locality

T_1 = service_time(pending_request)

T_2 = wait_time (future_request)

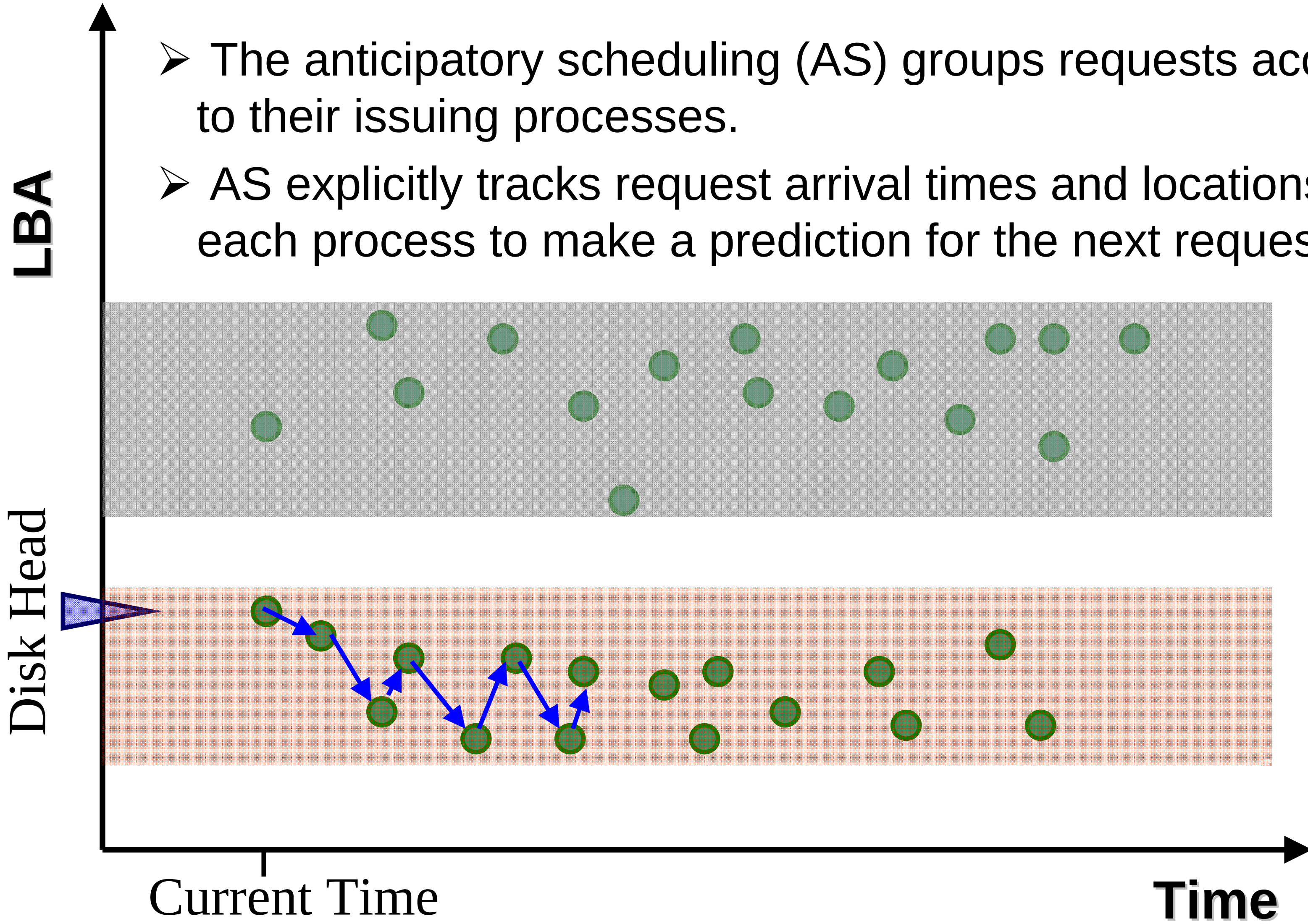
T_3 = service_time (future_request)



- Predicting arrival times and locations of future requests whose $T_2+T_3 < T_1$;
- Determining what request history should be used for the prediction.

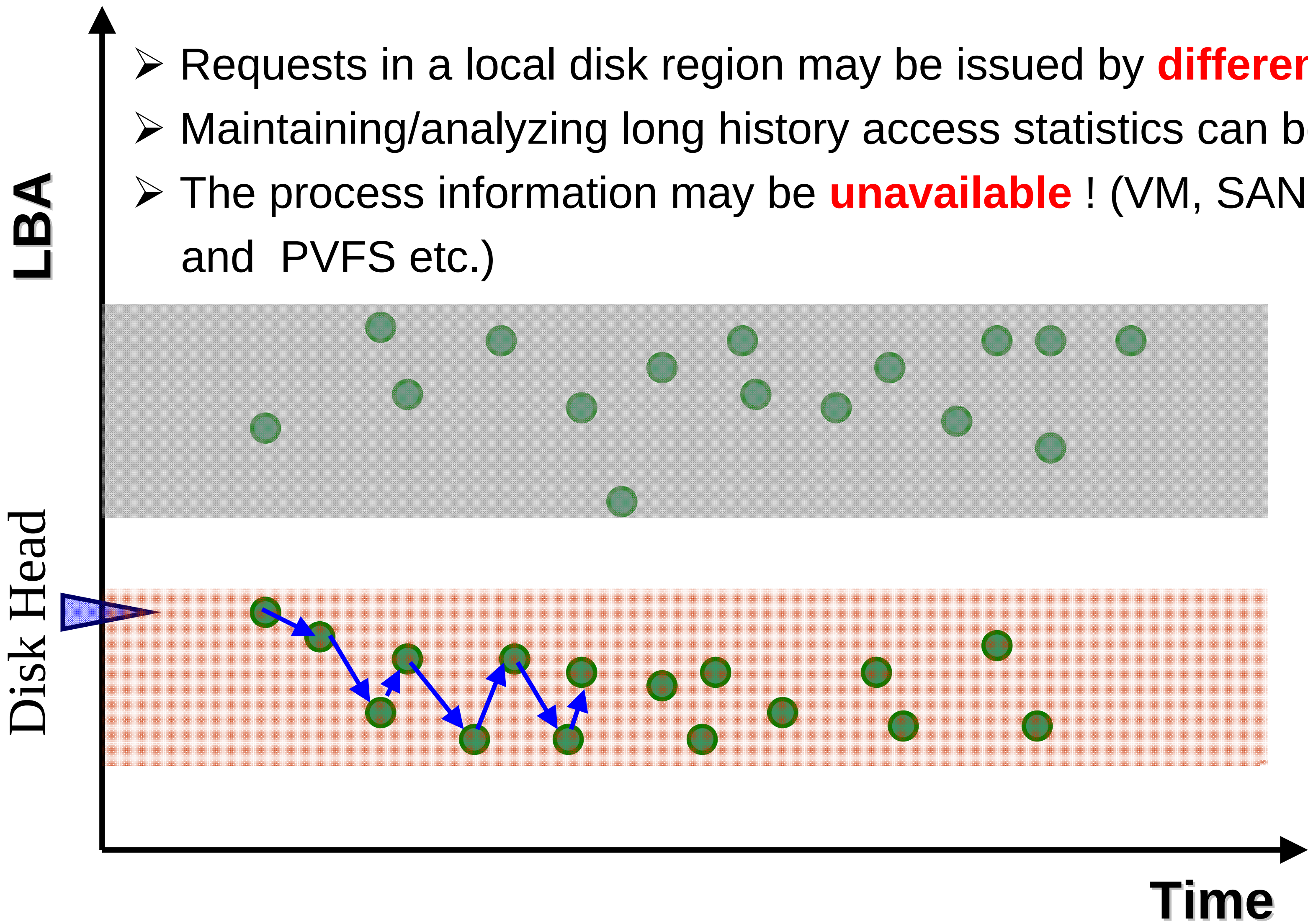
How does *anticipatory* handle them?

- The anticipatory scheduling (AS) groups requests according to their issuing processes.
- AS explicitly tracks request arrival times and locations for each process to make a prediction for the next request.



***Anticipatory*'s Limitations**

- Requests in a local disk region may be issued by **different** processes.
- Maintaining/analyzing long history access statistics can be **expensive**.
- The process information may be **unavailable** ! (VM, SAN, NFS, and PVFS etc.)



Related Approaches

- *Antfarm* infers process information in the virtual machine monitor by tracking activities of processes in VMs [*USENIX ATC'06*].
  - Applicable only to VM.
  - Guest OS needs to be open for instrumentation.

- Hints, such as accessed files' directory or owner, are used for grouping requests in the NFS servers. [*Cluster'08*].
  - Hints may not be always relevant.

- The Linux prefetching policy exploits spatial locality by tracking file access for every processes' opened file. [*Linux Symposium'04]*
  - File abstraction may not be available to the disk schedulers.
  - Its efficient tracking and decision making mechanisms can be leveraged.

Design Goals of Stream Scheduling

- Use only request characteristics, i.e., request arrival times and locations
  - Process information is not required in any way.
- Introduce minimal overhead
  - Remember minimal history access information
  - Conduct minimal computation in its locality analysis
- Integrate seamlessly with any work-conserving schedulers
  - Designed as a framework to make them non-work-conserving

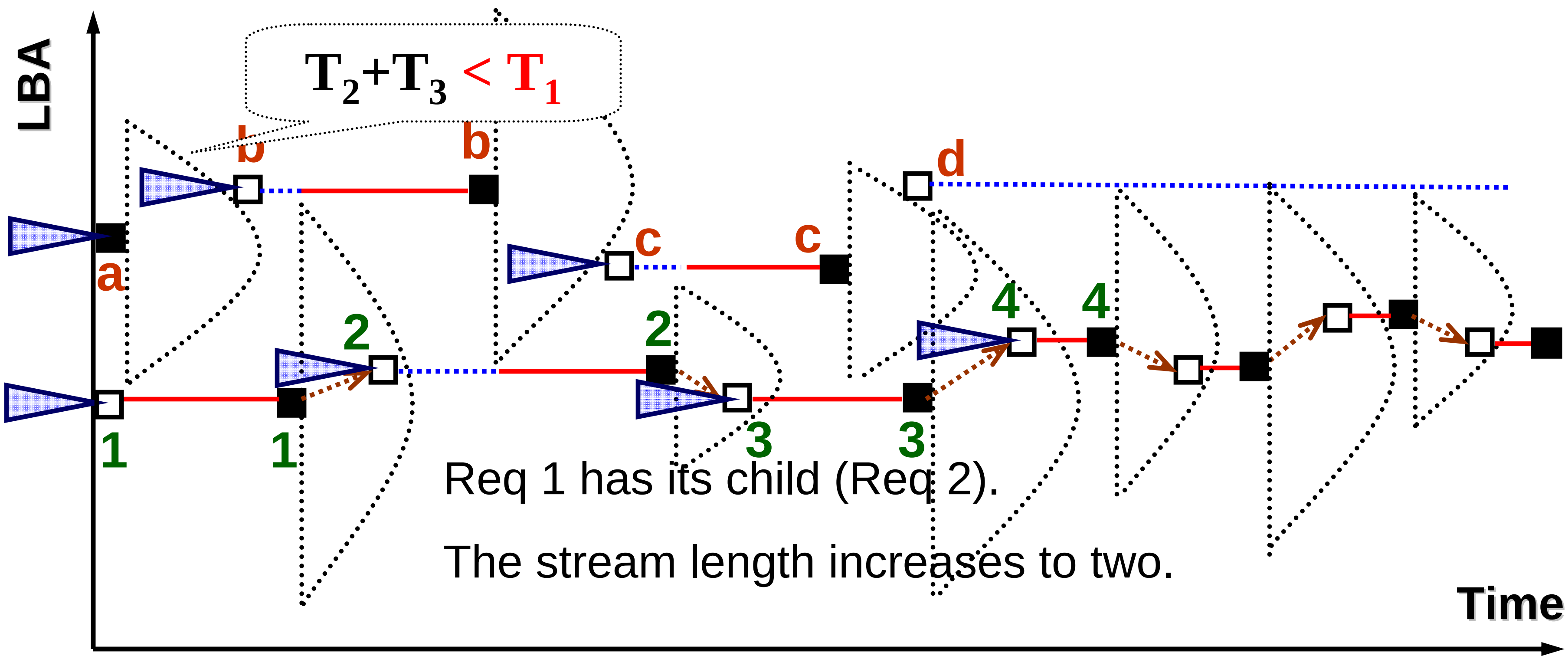
Design of Stream Scheduling

- Group requests into streams so that the intra-stream locality is stronger than the inter-stream locality.

- Track judicious scheduling decisions rather than locality metrics
  - Wait or not wait? (future request vs. pending request)
  - A stream is a sequence of requests for which judicious decisions are "wait".

- A stream is maintained as Linux prefetching does.
  - A stream is built up or torn down depending on next judicious decision.

Stream Scheduling Illustration



Arrival of a request

Completion of a request

Time period serving other requests

Time period serving this request

Link showing relationship between parent request and child request

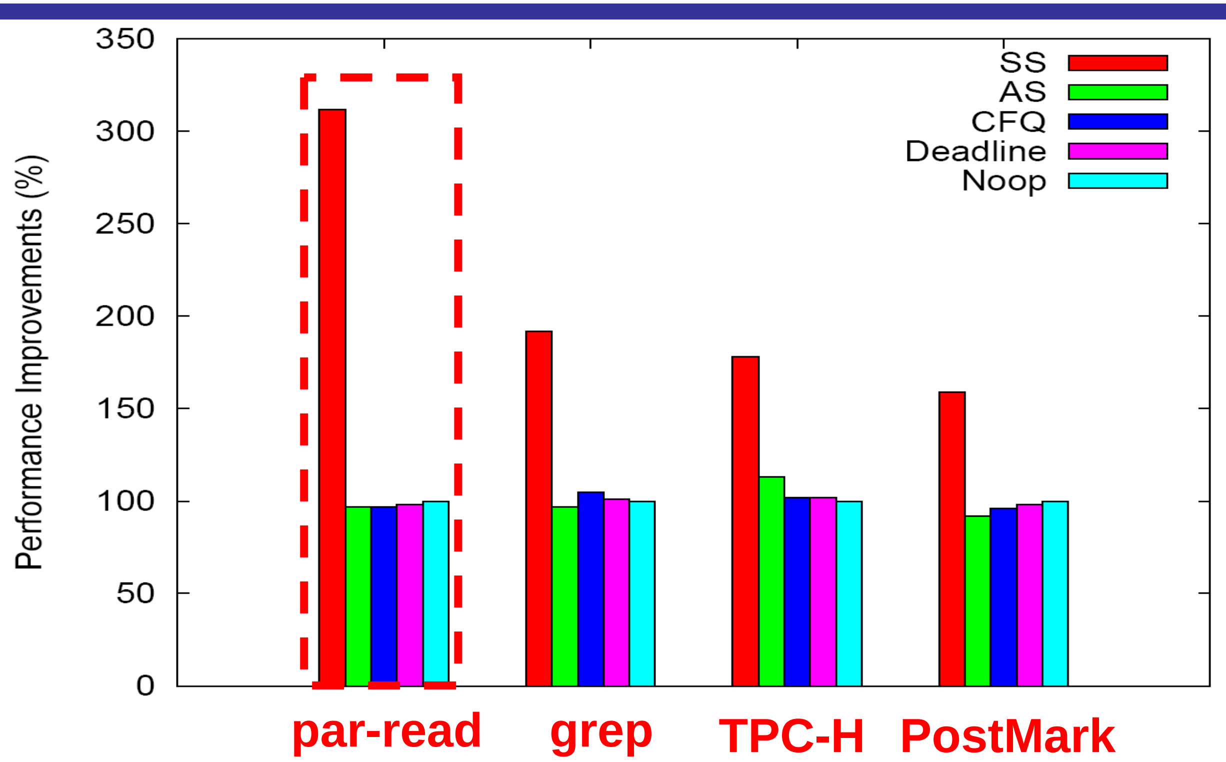
Maintenance of Streams

- A stream grows when a completed request sees its child.
  - Determining existence of a child is independent of actual scheduling.
  - A stream is established when its length exceeds a threshold.
  - An established stream leads to non-work-conserving scheduling.
- The scheduler stops serving a stream when
  - the stream is broken; or
  - the time slice allocated to the stream runs out; or
  - an urgent request appears.
- To maintain a stream, only current stream lengths need to be remembered.
  - The cost is trivial !
- We have design of stream scheduling for the disk array.
  - It is described in the paper.

Experiment Settings

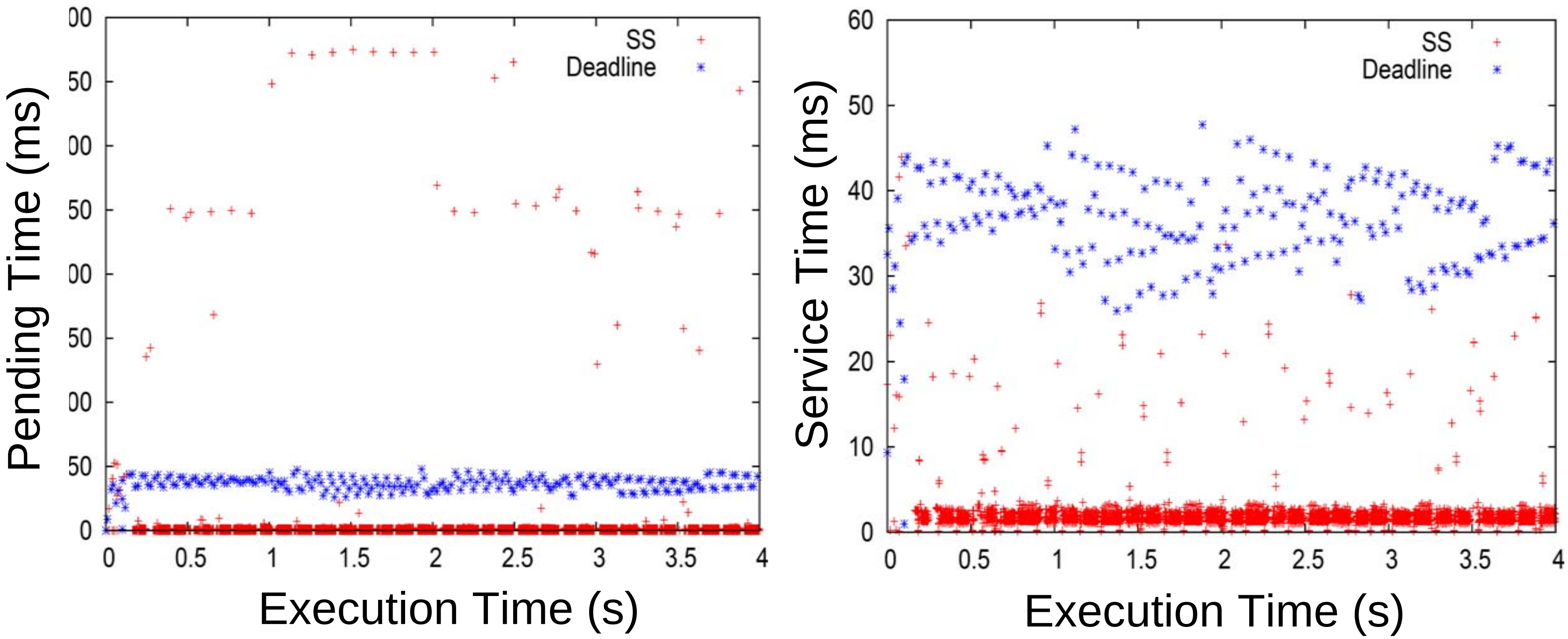
- Software settings
  - Stream Scheduling (SS) is prototyped in **Linux kernel 2.6.31.3** using **Deadline** as its work-conserving component.
  - The default stream length threshold is **4**.
  - The default stream time slice is **124ms**.
- Hardware settings
  - Intel Core2 Duo with 2GB DRAM memory.
  - 7200RPM, 500GB Western Digital Caviar Blue SATA II with a 16MB built-in cache.
- Adaptation for **NCQ**
  - Disk head position is indicated by the last request sent to the disk.

Storage without Process Information



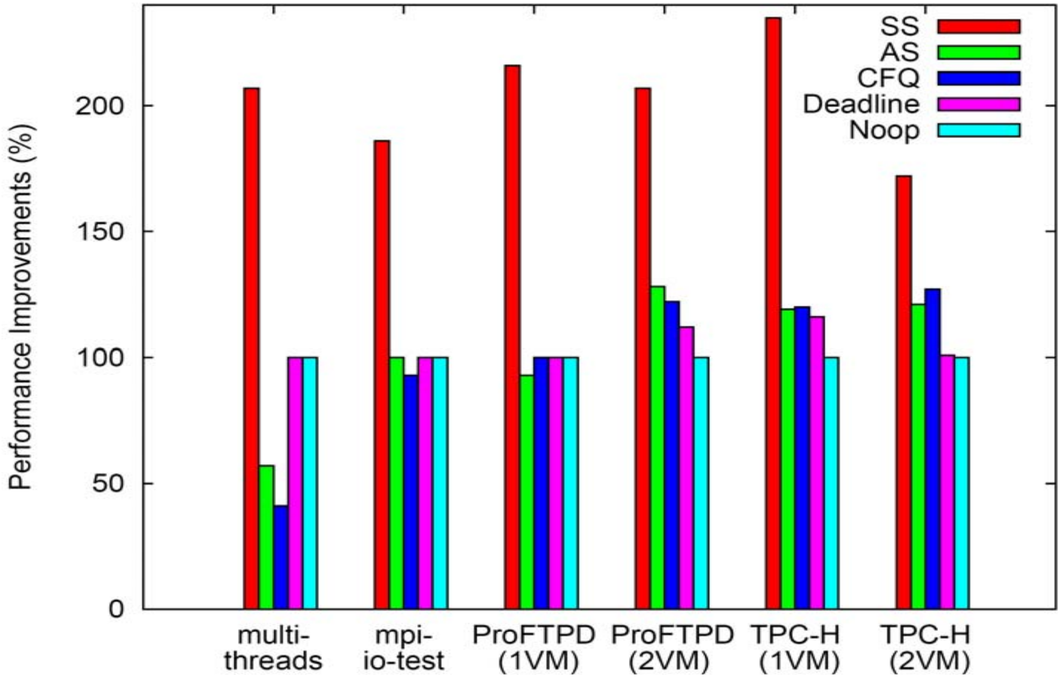
- ***par-read***: four independent processes, each reading a 1GB file using 4KB requests in parallel.
- ***Grep***: two *grep* instances, each searching in a Linux directory tree.
- ***TPC-H***: three TPC-H instances, each using PostgreSQL as its database server and DBT3 to create its tables.
- ***PostMark***: four PostMark instances, each creating a data set of 10,000 files.

Storage without Process Information



par-read: four independent processes, each reading a 1GB file using 4KB requests in parallel.

Storage with Inadequate Process Information



- ***multi-threads:*** four processes, each forking two threads for reading files with periodic synchronization between them.
- ***mpi-io-test:*** four *mpi-io-test* program instances running on PVFS2 where files are striped over eight data servers.
- ***ProFTPD:*** a *ProFTPD* FTP server on each Xen VM supporting four clients to simultaneously download four 300MB files.
- ***TPC-H:*** three *TPC-H* instances on each Xen VM.

Conclusions

- The stream scheduling framework turns any disk scheduler into a non-work-conserving one.
  - Process information is not required in the scheduling.
  - Both time and space overheads are low.
- The framework can be extended to disk arrays to recover and exploit the locality weakened by file striping.
- Experiments on its Linux prototype show significantly improved performance for representative benchmarks.