Anticipatory Schuduling
:A disk scheduling framework to overcome deceptive idleness in synchronous I/O

Kim Sang Eon
July 26, 2010
SUNG KYUN KWAN Univ.
SAMSUNG ELECTRONICS CO., LTD.

Title

- Anticipatory Scheduling
  : A disk scheduling framework to overcome deceptive idleness in synchronous I/O

- anticipatory
  예상 하고서, 예상하고 앞질러 하는
  선행하는

- Deceptive idleness
  기만하는, 현혹하는 게으름

- Synchronous I/O
  동기식 I/O ex) read
  ↔ asynchronous ex) write

Why I/O scheduler

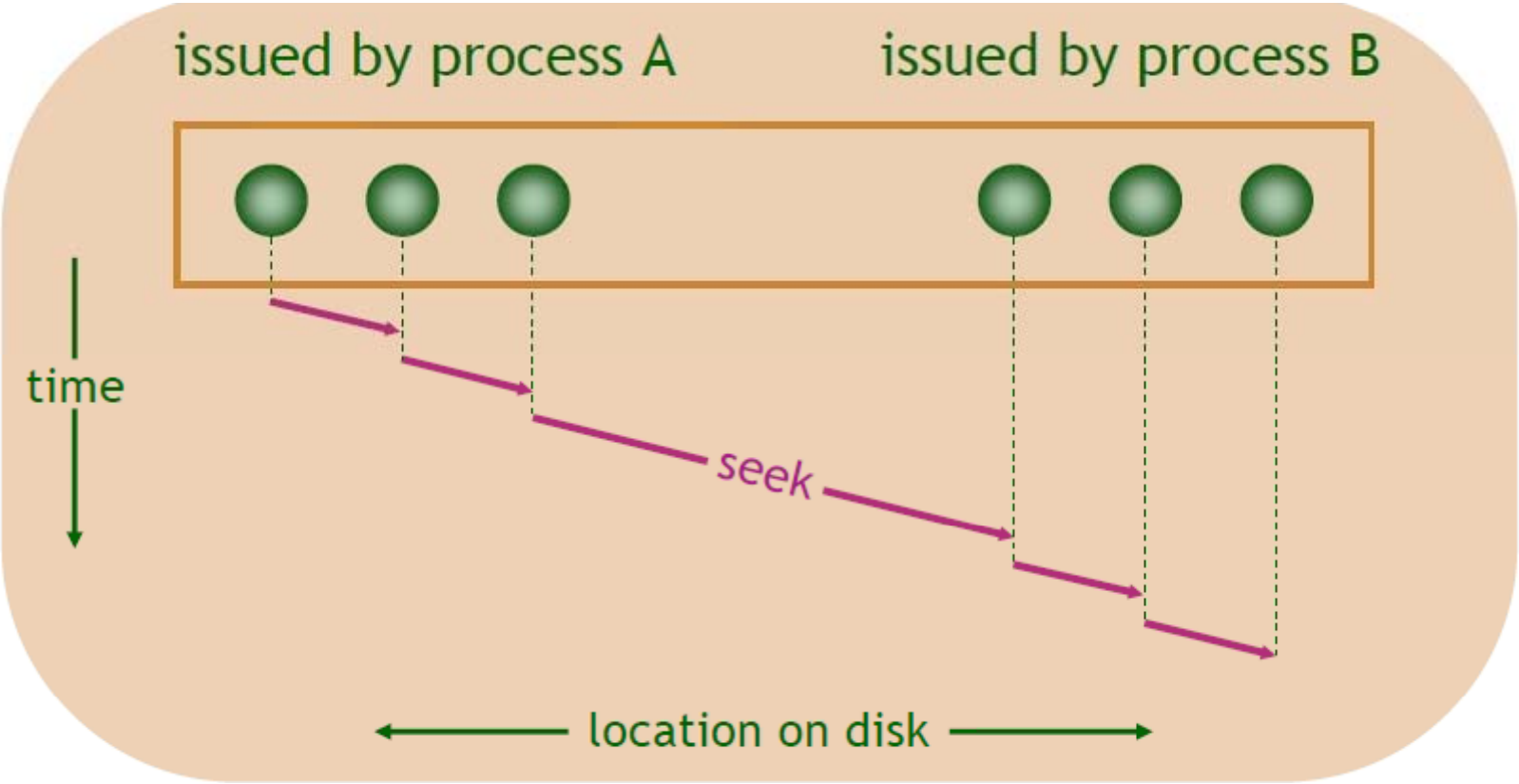
- Disk seek is the slowest operation in a computer
  - ✓ A system would perform horribly without a suitable I/O scheduler
- Improve overall disk throughput by
  - ✓ Reorder request to reduce the disk seek time
  - ✓ Merge request to reduce the number of requests
- Prevent starvation
  - ✓ Submit request before deadline
  - ✓ Avoid read starvation by write
- Provide fairness among different processes

Anticipatory scheduling Background

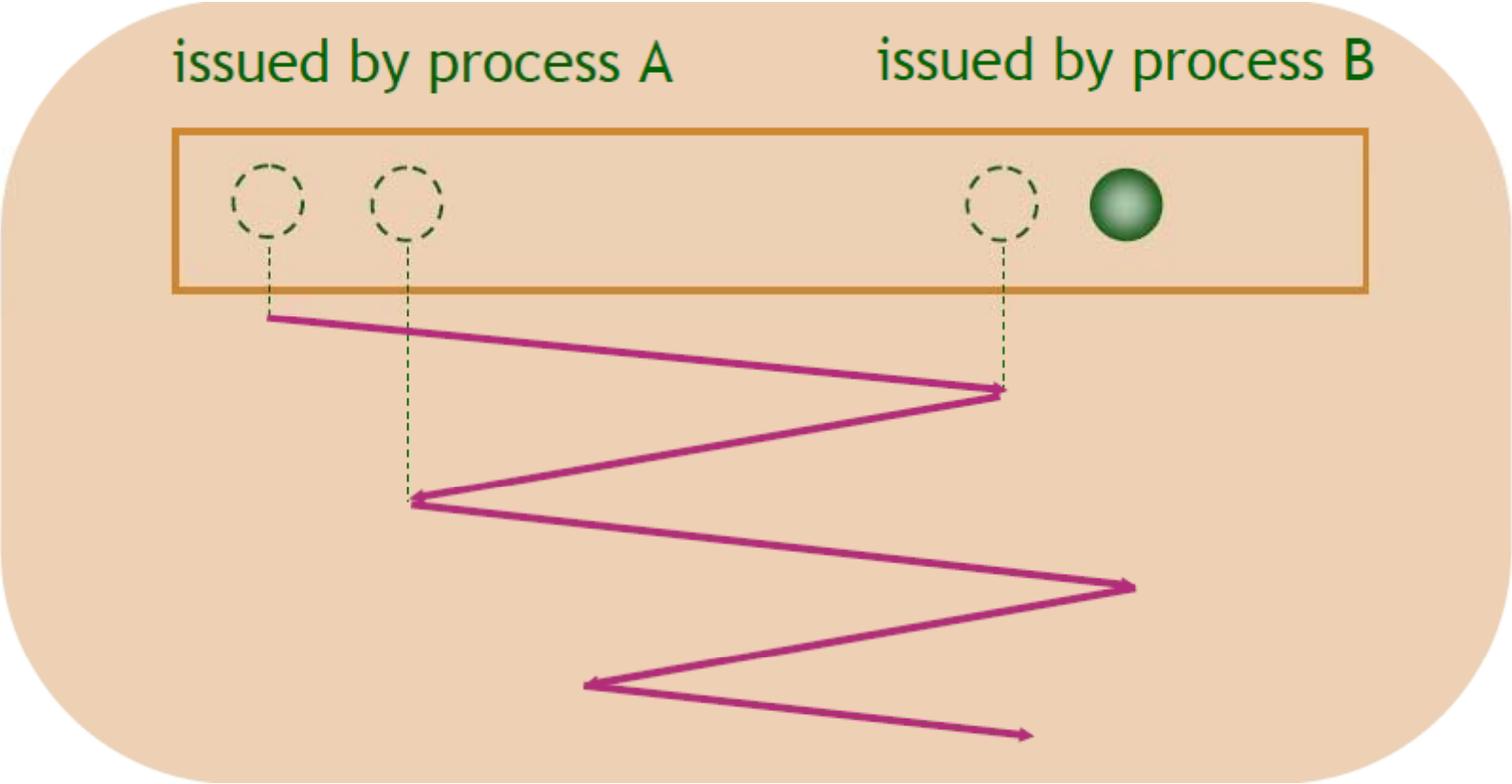
- Disk Schedulers reorder available disk request for
  - ✓ performance by seek optimication
  - ✓ proportional resource alloccation

- Any policy needs multiple outstanding request to make good decisions!

With enough request



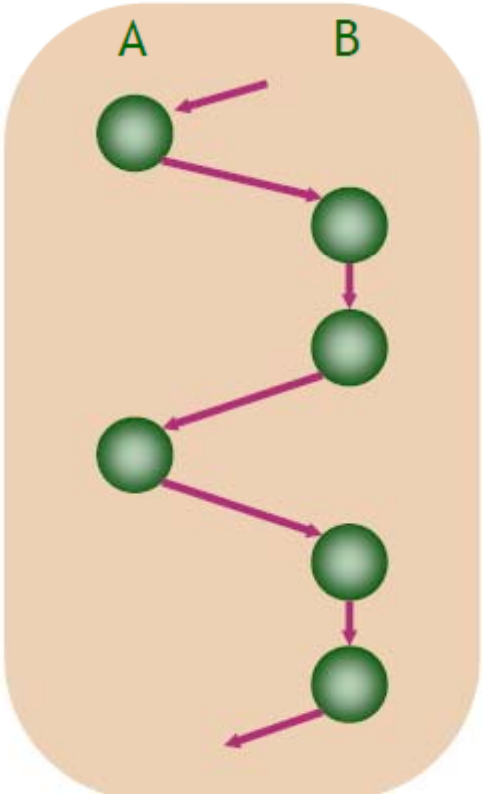
With Synchronous I/O



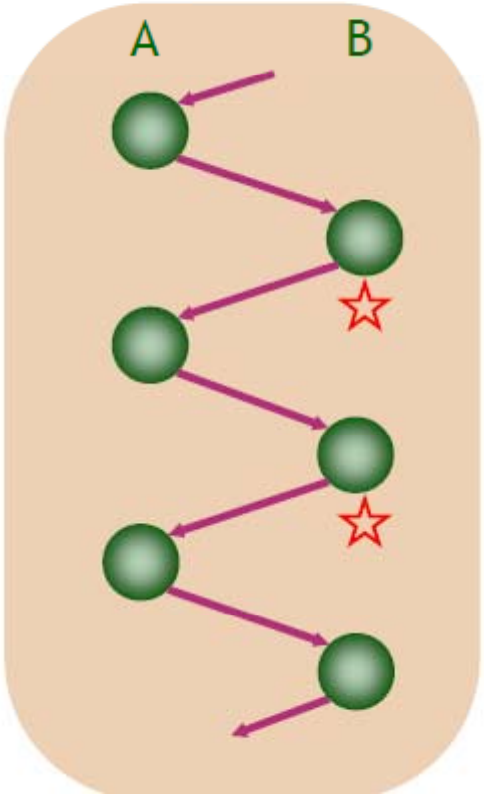
- Deceptive Idleness
  - ✓ Process A is about to issue next request
  - ✓ But scheduler hastily assumes that process A has no furder request

Proportional scheduler

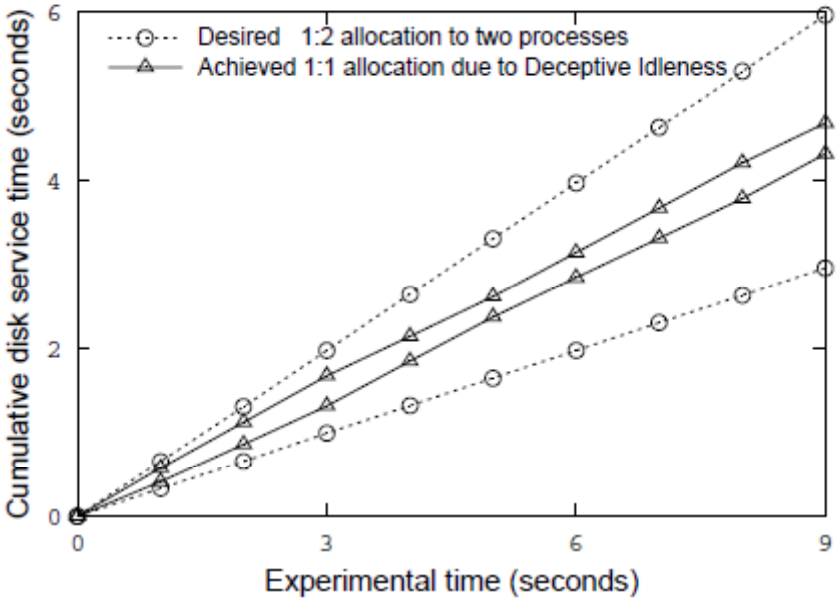
Allocate disk service in say 1:2 ratio:



Deceptive idleness causes 1:1 allocation:



Proportional scheduler



Prefetching

- Asynchronous prefetch
  - ✓ Partially work around
  - ✓ Predicting the future request issue pattern
  - ✓ Limitation

  - ✓ Application-driven prefetch
    - Incapable of issuing accurate prefetch request
    - Have to be rewritten
    - More data copying and cache pollution
    - Not enabled in some operating systems
  - ✓ Kernel-driven prefetch
    - Less capable of predicting future access patterns
    - Penalties of misprediction can be high

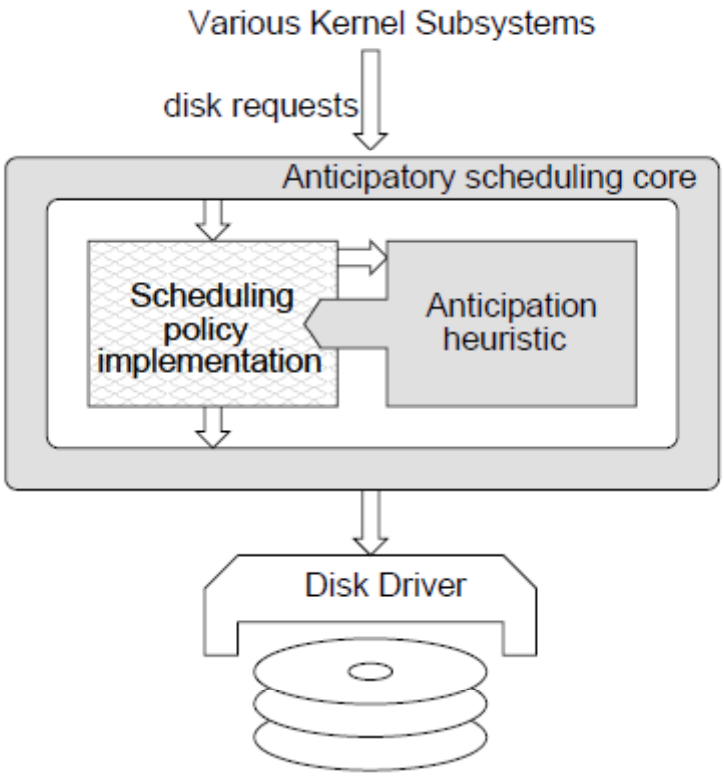
Anticipatory Scheduling

- Key idea
  - ✓ Briefly wait for process whose request was last serviced
- Work-conserving
  - ✓ If the disk is idle or a request is completed,
  - ✓ Next request in the queue is scheduled immediately
- Non-work-conserving
  - ✓ The disk stands idle in the face of nonempty queue

How long to wait

- Benefit of waiting > Cost of keeping idle
- Need Heuristics !!

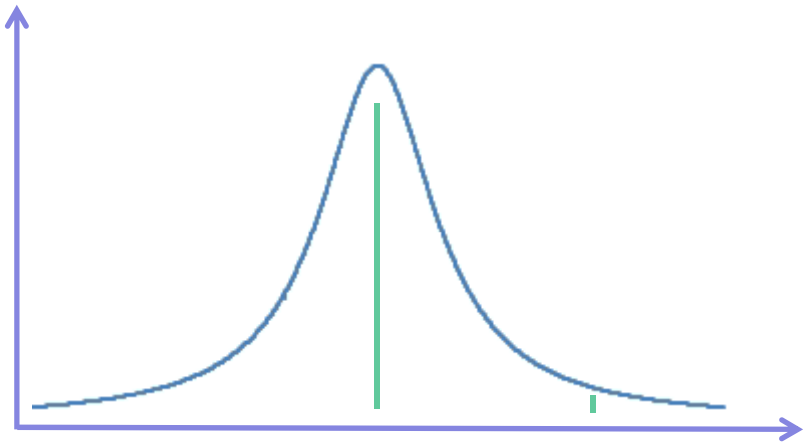
Anticipatory Scheduling Framework



Benefit-Cost Analysis

- Seek reducing scheduler

$$
\begin{aligned}
benefit &= (calculate\_positioning\_time(Candidate) \\
&\quad - \text{LP}.expected\_positioning\_time) \\
cost &= \max(0, \\
&\quad \text{LP}.expected\_median\_thinktime - elapsed) \\
waiting\_duration &= \max(0, \\
&\quad \text{LP}.expected\_95percentile\_thinktime - elapsed) \\
return\ &(benefit > cost\ ?\ waiting\_duration\ :\ 0)
\end{aligned}
$$

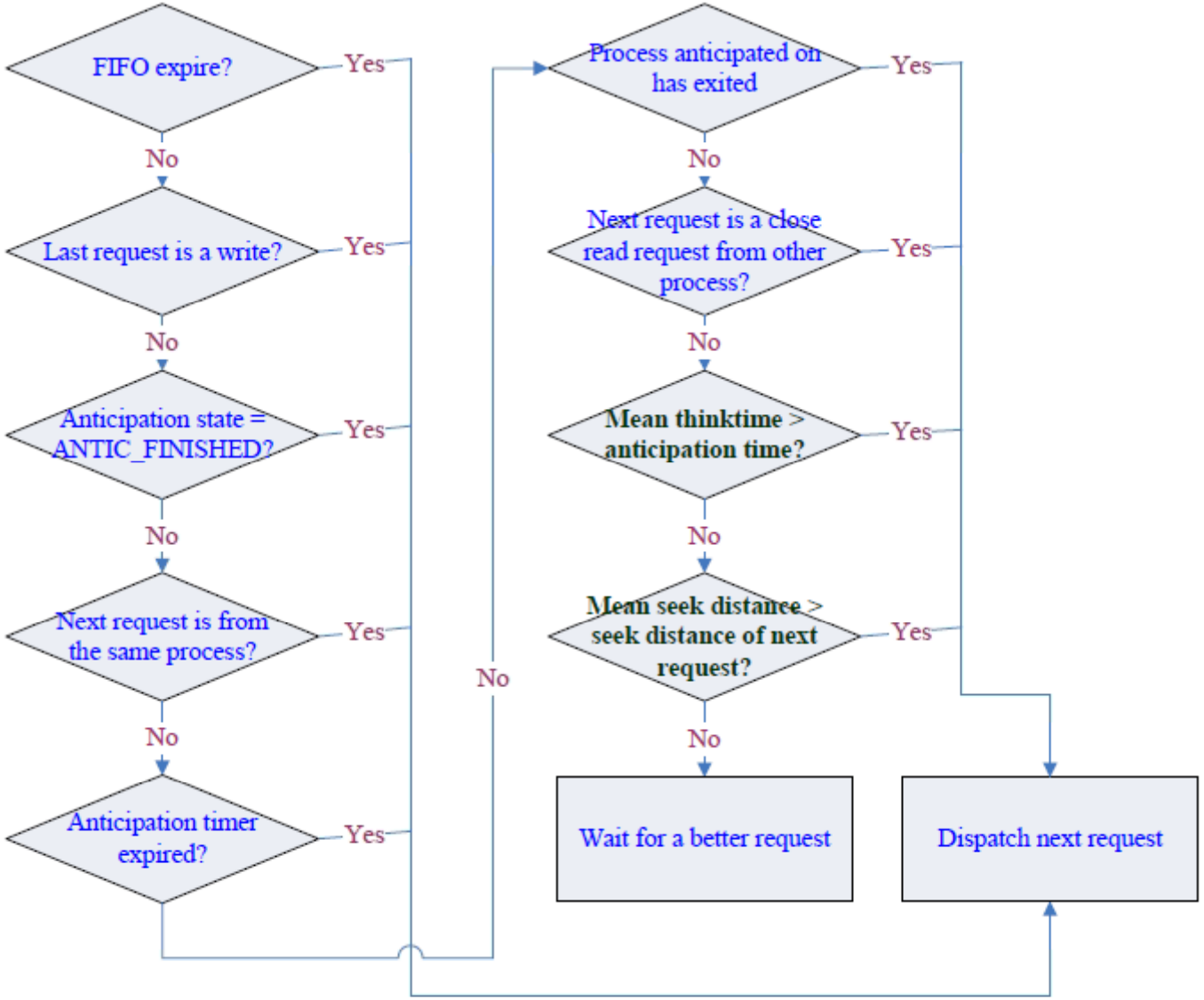


Benefit-Cost Analysis

- Proportional scheduler

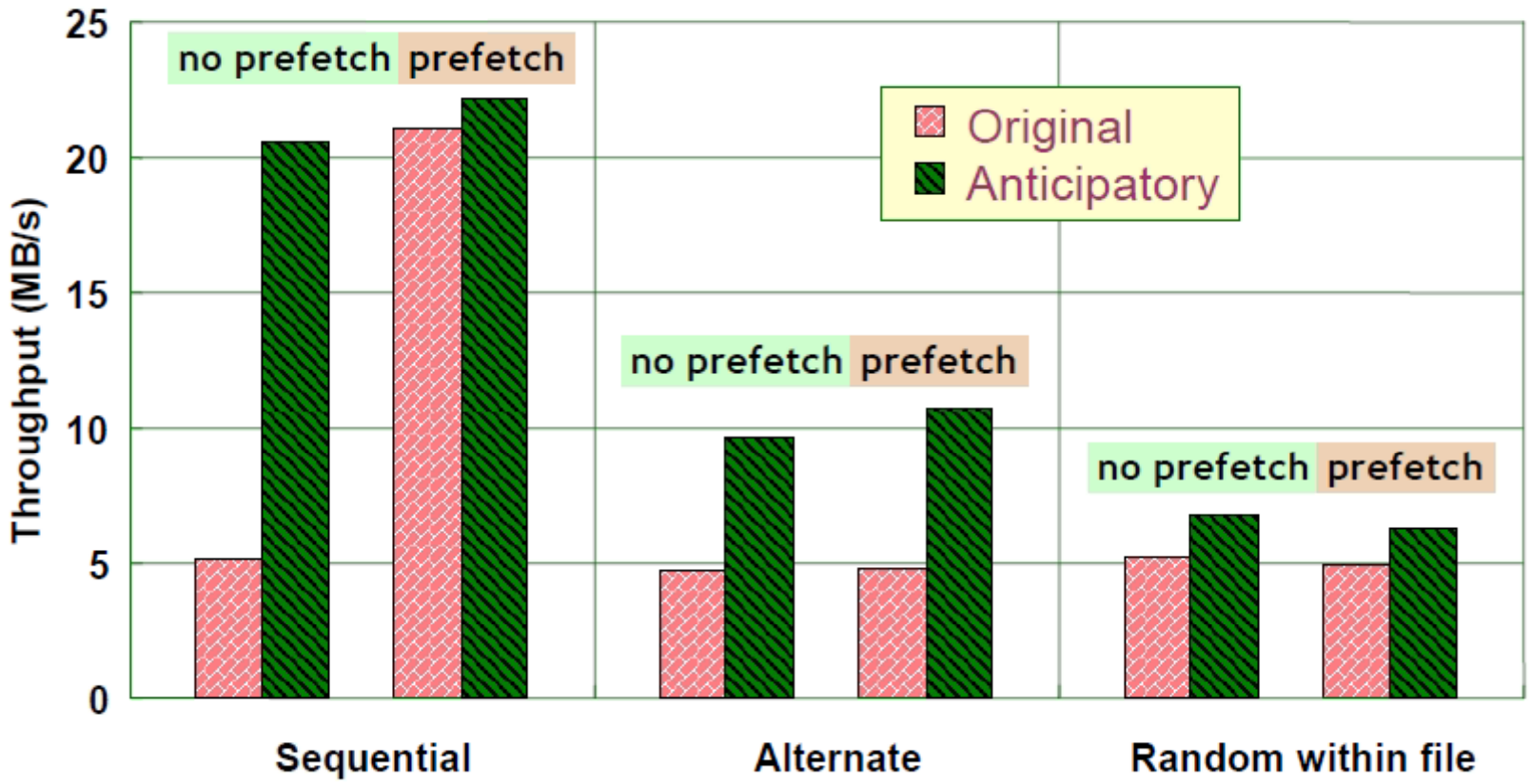
- Conditions of Waiting for process whose req was last serviced
  - ✓ If it has received less than its allocation
  - ✓ If it has thinktime below a threshold (ex. 3ms)

Flow chart


FIFO expire?
Yes
No
Last request is a write?
Yes
No
Anticipation state = ANTIC_FINISHED?
Yes
No
Next request is from the same process?
Yes
No
Anticipation timer expired?
Yes
No
Process anticipated on has exited
Yes
No
Next request is a close read request from other process?
Yes
No
Mean thinktime > anticipation time?
Yes
No
Mean seek distance > seek distance of next request?
Yes
No
Wait for a better request
Dispatch next request

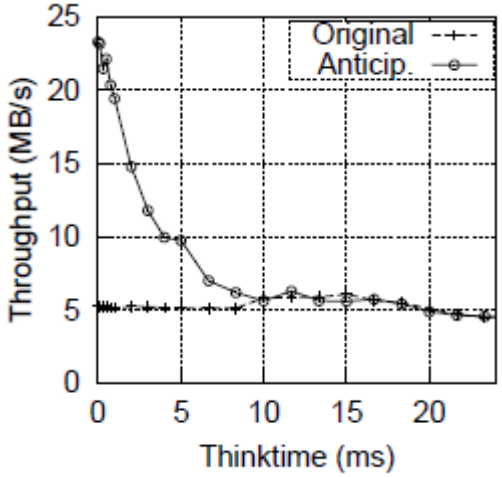
Evaluation

- Microbenchmark : Access patterns

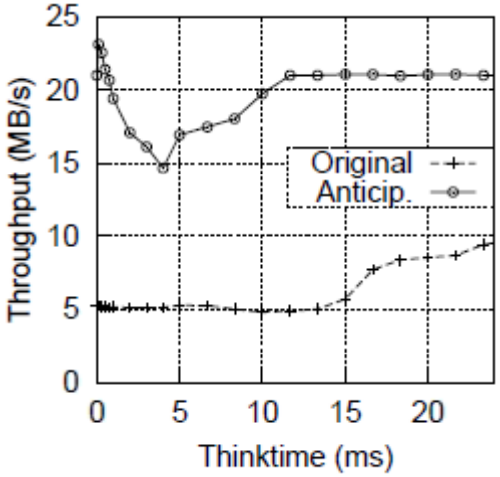


Evaluation

- Microbenchmark : Varying thinktimes



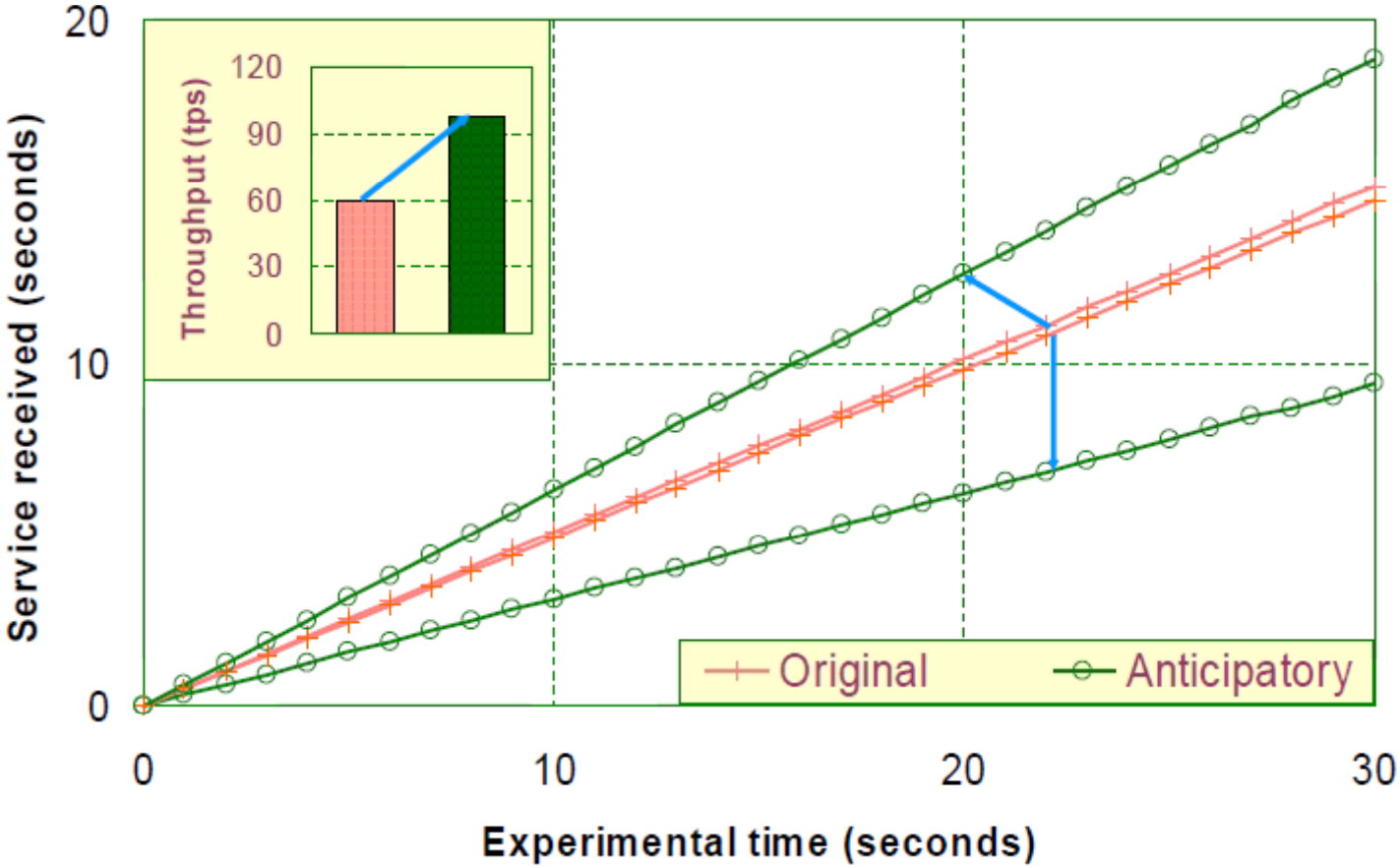
Symmetric process



Asymmetric process

Evaluation

- Proportional-share Scheduling



Cooperative Anticipatory Scheduler

- Proposed in this paper
  - ✓ Enhancements to Linux I/O scheduler, Seetharami Seelam, OLS2005
- The problems of anticipatory Assumption
  - ✓ Synchronous disk req are issued by individual processes
  - ✓ For anticipation to work properly, the anticipatedprocess must be alive
- The problems of anticipatory scheduler
  - ✓ Anticipation works only when req are issued by same process

The problems of anticipatory scheduler

- Concurrent streaming and chunk read programs.
  - ✓ By assume 1, problem occure
  - ✓ Program A : synchronous read request by a single process
  - ✓ Program B : dependent chunk read by a different process
    ex) shell script
  - ✓ Anticipation works only on a per-process basis (Program A)
  - ✓ Concurrent execution of these two programs results in starvation of processes generated by Program B.

The problems of anticipatory scheduler

- Concurrent chunk read programs.
  - ✓ By assume 2, problem occure
  - ✓ Program B : dependent chunk read by a different process
  - ✓ Process1 and Process2 will work well.
  - ✓ Once a process1 terminates (with pending request)
  - ✓ Dispatch a request of the Process2

  - ✓ If there are F files, at least 2F-1 seeks are necessary
  - ✓ Seek storms. Poor throughput

CAS solution

- Solution
  - ✓ Keep anticipating even when the anticipated process has exited
  - ✓ Cooperative exit probability
  - ✓ Existence of cooperative processes related to dead processes

CAS Evaluation

- Streaming and chunk reads
  - ✓ Anticipation works well for A
  - ✓ But, what happens with B
  - ✓ B:inadmissable time using AS
  - ✓ CAS provides anticipation on a per-group basis; thus seeks reduced and throughput improved

Program A:

```
while true
do
   cat big-file > /dev/null
done
```

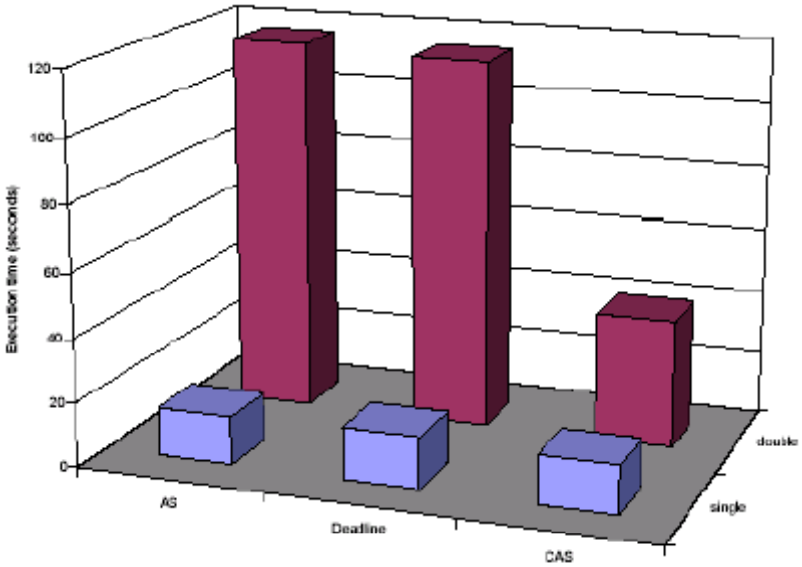
Program B:

```
time find . -type f -exec \
        cat '{}' ';' > /dev/null
```

Scheduler	Execution Time (sec.)	Throughput (MB/s)
Deadline	297	9
LAS	4767	35
CAS	255	34

CAS Evaluation

- Multiple chunk reads
  - ✓ Two instances of chunk read to disjoint disk blocks
  - ✓ Anticipation fail to both
  - ✓ Results for reading Linux source tree
  - ✓ There is some seeking, but CAS does not seek as much as others



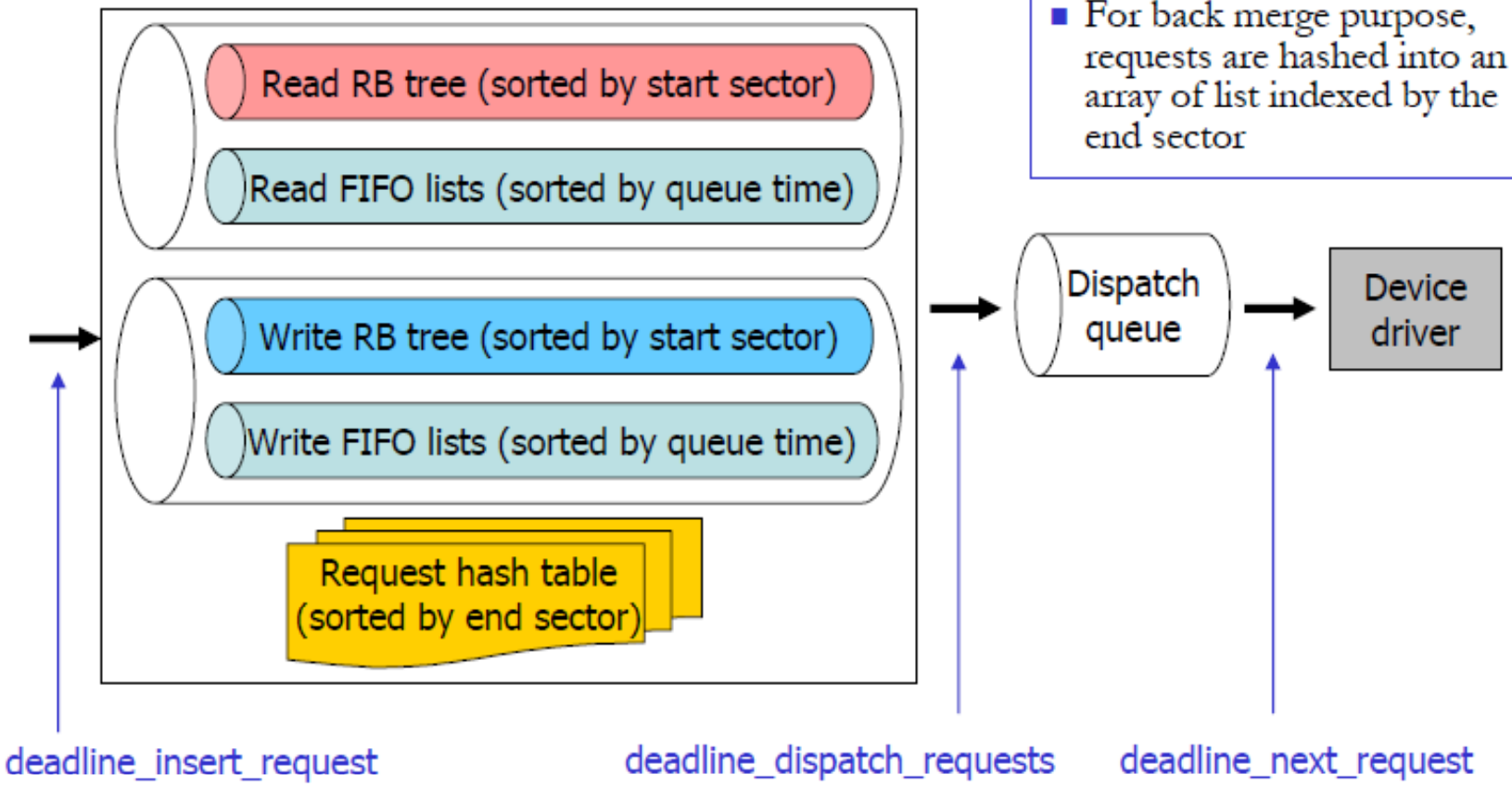
Scheduler	Throughput (MB/s)	
	1 Instance	2 Instances
Deadline	14.5	4.0
LAS	15.5	4.0
CAS	15.5	11.6

Deadline I/O scheduler

- Goal
  - ✓ Reorder request to improve I/O performance while simultaneously ensuring that no I/O request is being starved
- Each request is associated with a expire time.
  - ✓ Read:500ms, Write:5sec
- Request are inserted into
  - ✓ A sorted-by-start-sector queue (two queues, for read and write)
  - ✓ A FIFO list (two list) sorted by expire time
- Normally, request are pulled from sorted queues. However, if the request at the head of either FIFO queue expires, requeset are still processed in sorted order but started from the first request in the FIFO queue

Architecture of Deadline I/O scheduler

- The sorted queues are built on red-black trees
- For back merge purpose, requests are hashed into an array of list indexed by the end sector



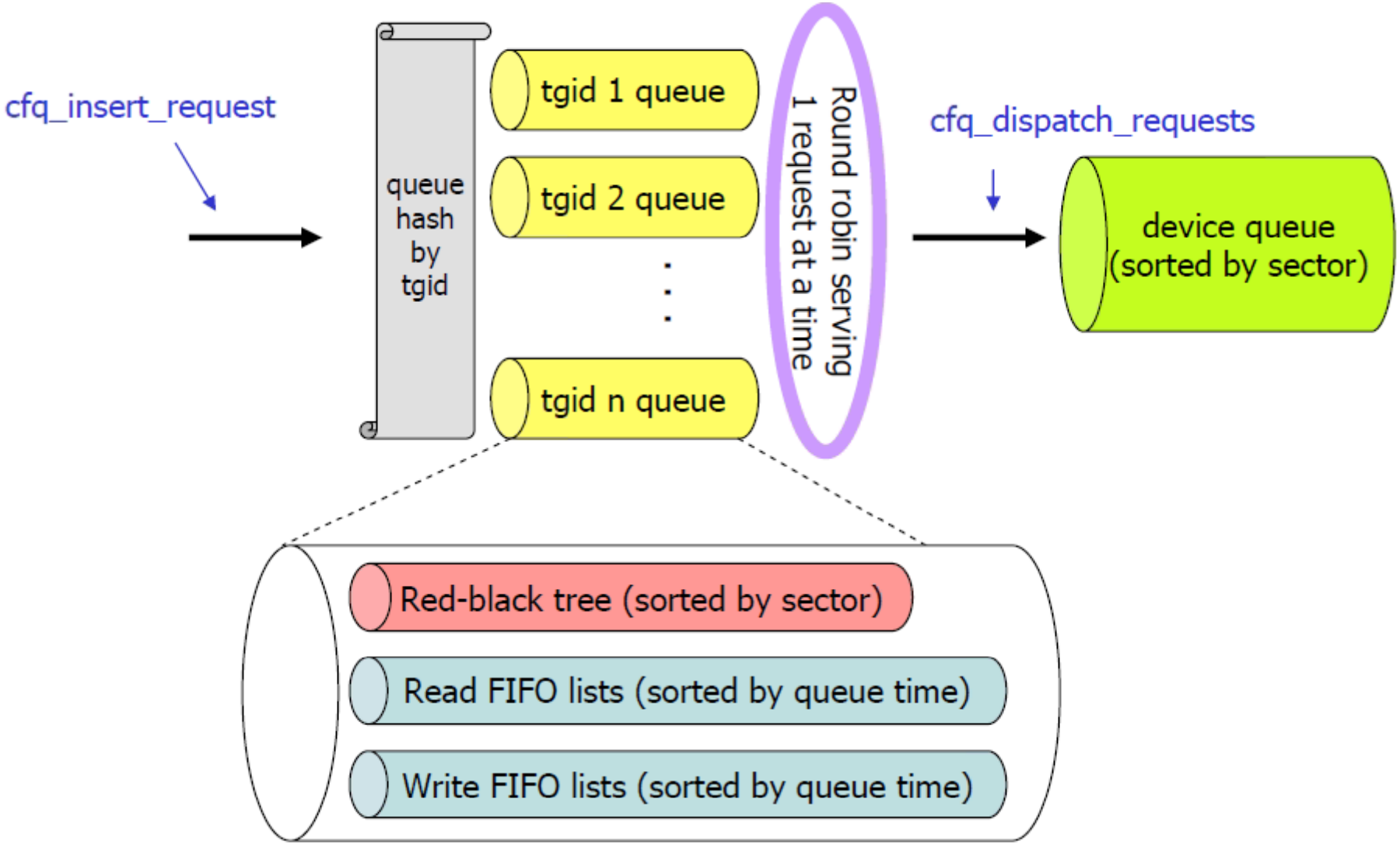
CFQ(Complete Fair Queuing)

- Goal
  - ✓ Provide fair allocatoin of I/O bandwidth among all the initiators of I/O request

- CFQ can be configured to provide fairness at per process, per process group, per user and per user group levels.

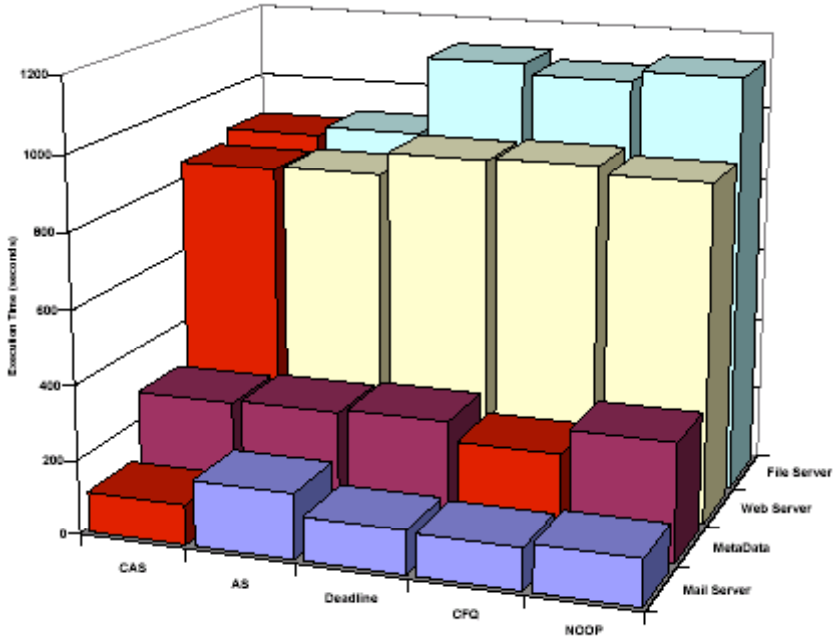
- Each initiator has its own request queue and CFQ services these queues round-robin

Architecture of CFQ



Evaluation of each scheduler

- AS has execution time comparable to CAS in Web server
- CAS has best in Mail server
- CAS has best in File server
- CFQ has best in Meta data



Reference

- Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O, SitaramIyer, ACM SOSP'01

- Enhancements to Linux I/O scheduling, SeetharamiSeelam, OLS'05

- *Research.utep.edu/Portals/937/Seelam**Linux**Symposium2005.pdf*