

# The Active Block I/O Scheduling System (ABISS)

Benno van den Brink

Philips Research, Eindhoven, The Netherlands

Werner Almesberger

Buenos Aires, Argentina

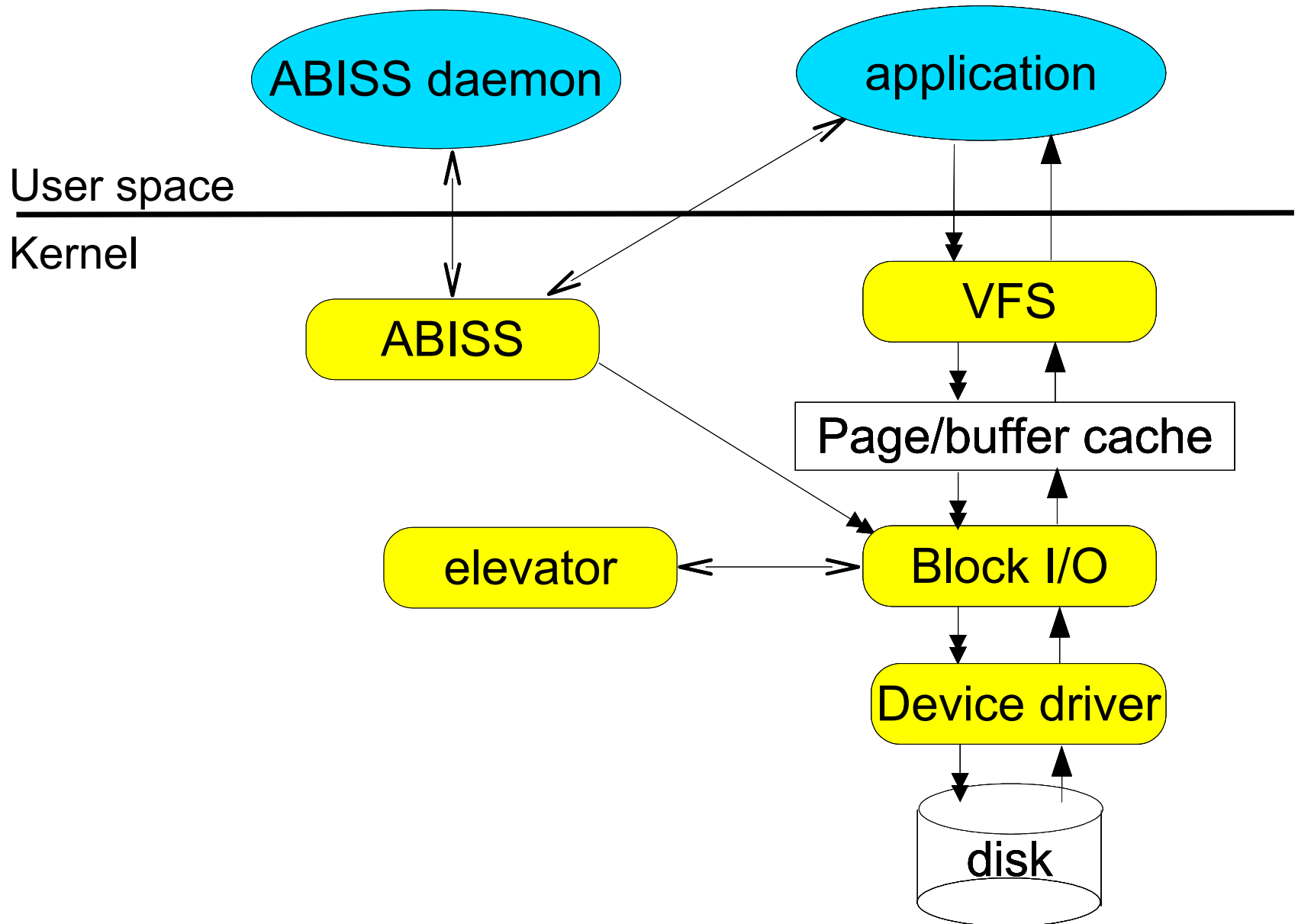
# ABISS

- Extension to Storage subsystem
- Allow applications to *prioritize* their I/O
- Provides a guaranteed hard disk I/O
  - Make sure the data is there when the application needs it
  - Provide system-wide control
  - Make better use of available performance
- Outline
  - Introduction and Overview
  - Implementation
  - Measurements
  - Future work and Conclusions

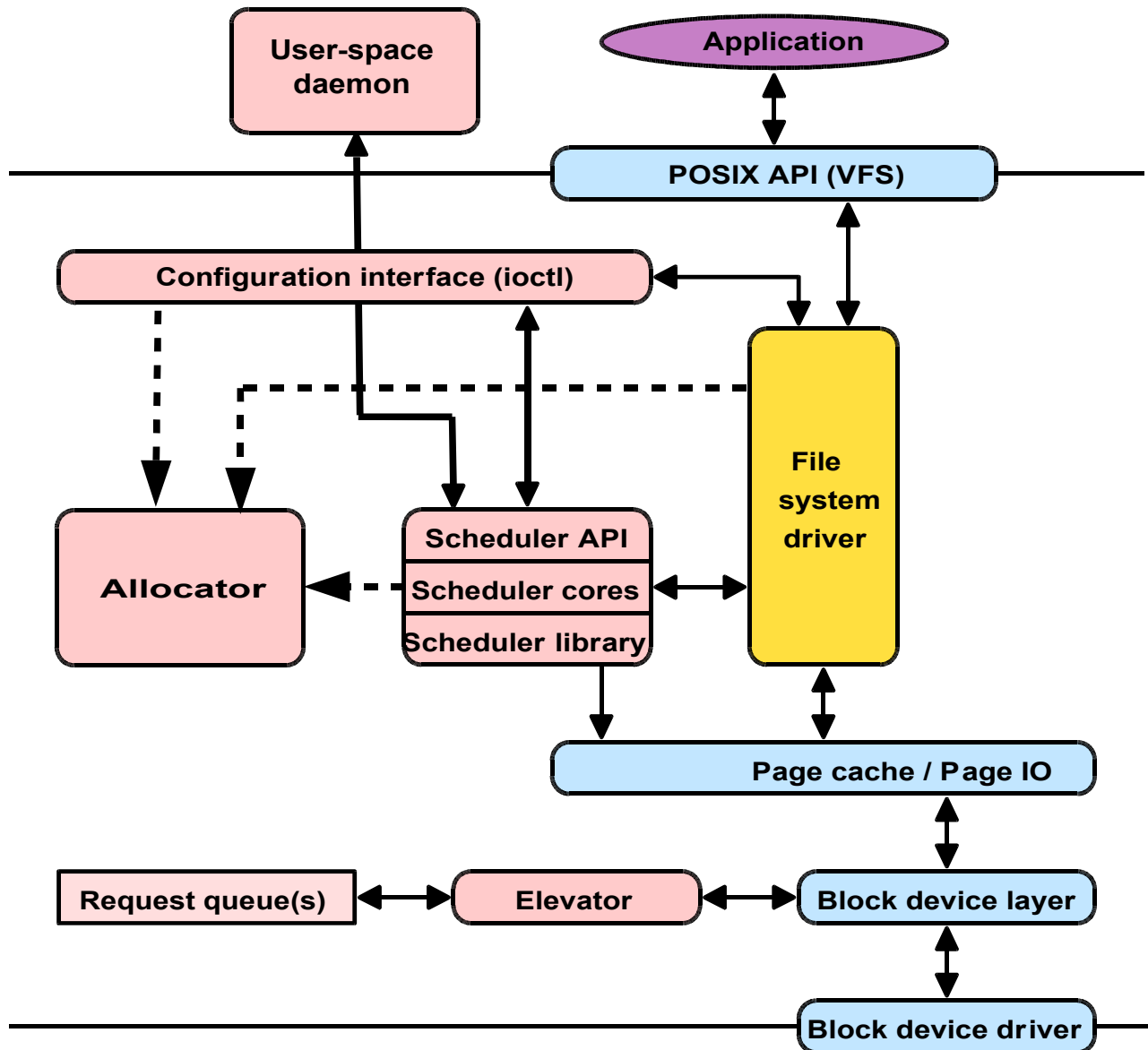
# Introduction

- HDDs in CE equipment
- Real-time streams: hick-ups are not acceptable
  - Multiple streams
- 'PC world' solution: Best Effort + performance overkill
  - Large buffers in application (cost, latency)
  - Performance (bandwidth, CPU) much higher than average need – over-dimension your system
  - Cost, power consumption, ...
- Just-in-time: make system guarantee bitrate for RT streams
- Handle BE traffic in remaining time (make sure there is!)

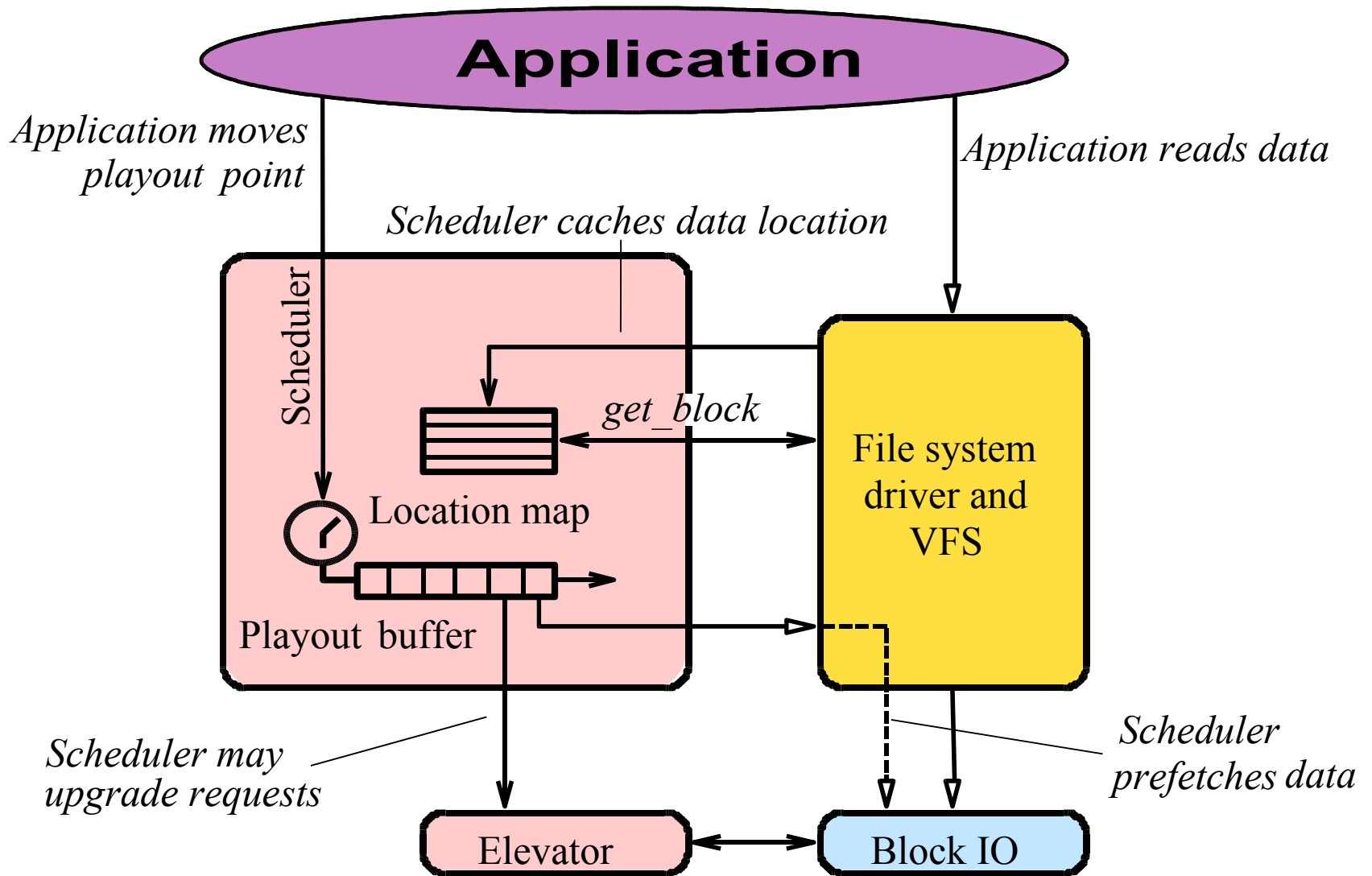
# Overall architecture



# implementation

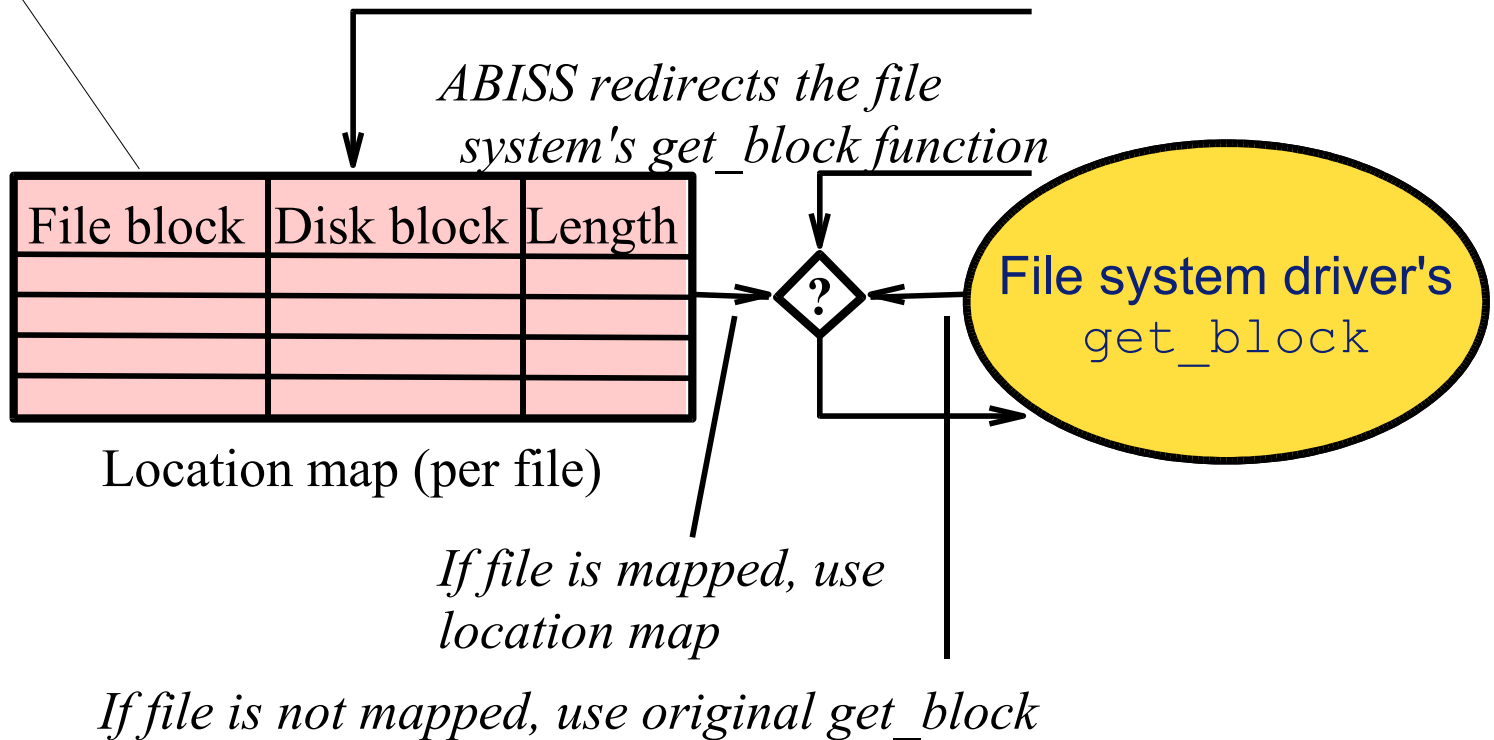


# scheduler

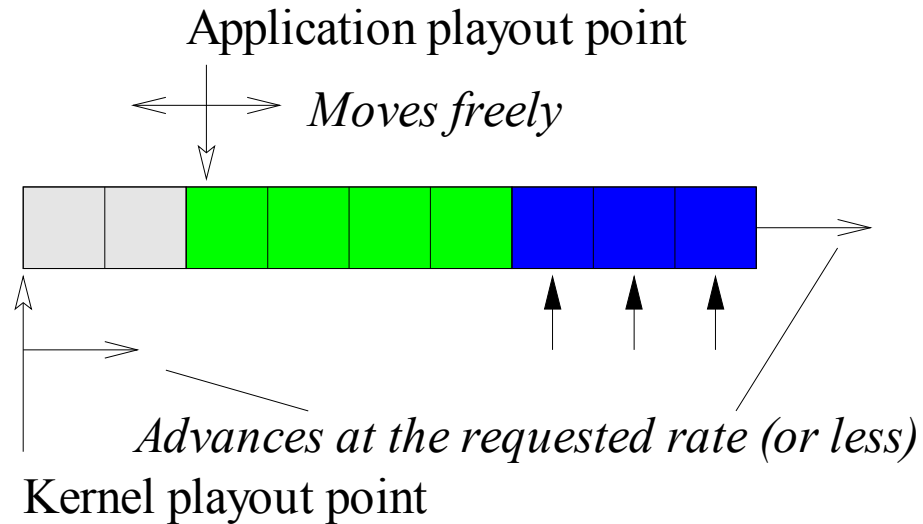


# Reading block position

store on-disk location  
of RT files



# Playout buffer



□ Page is no longer used

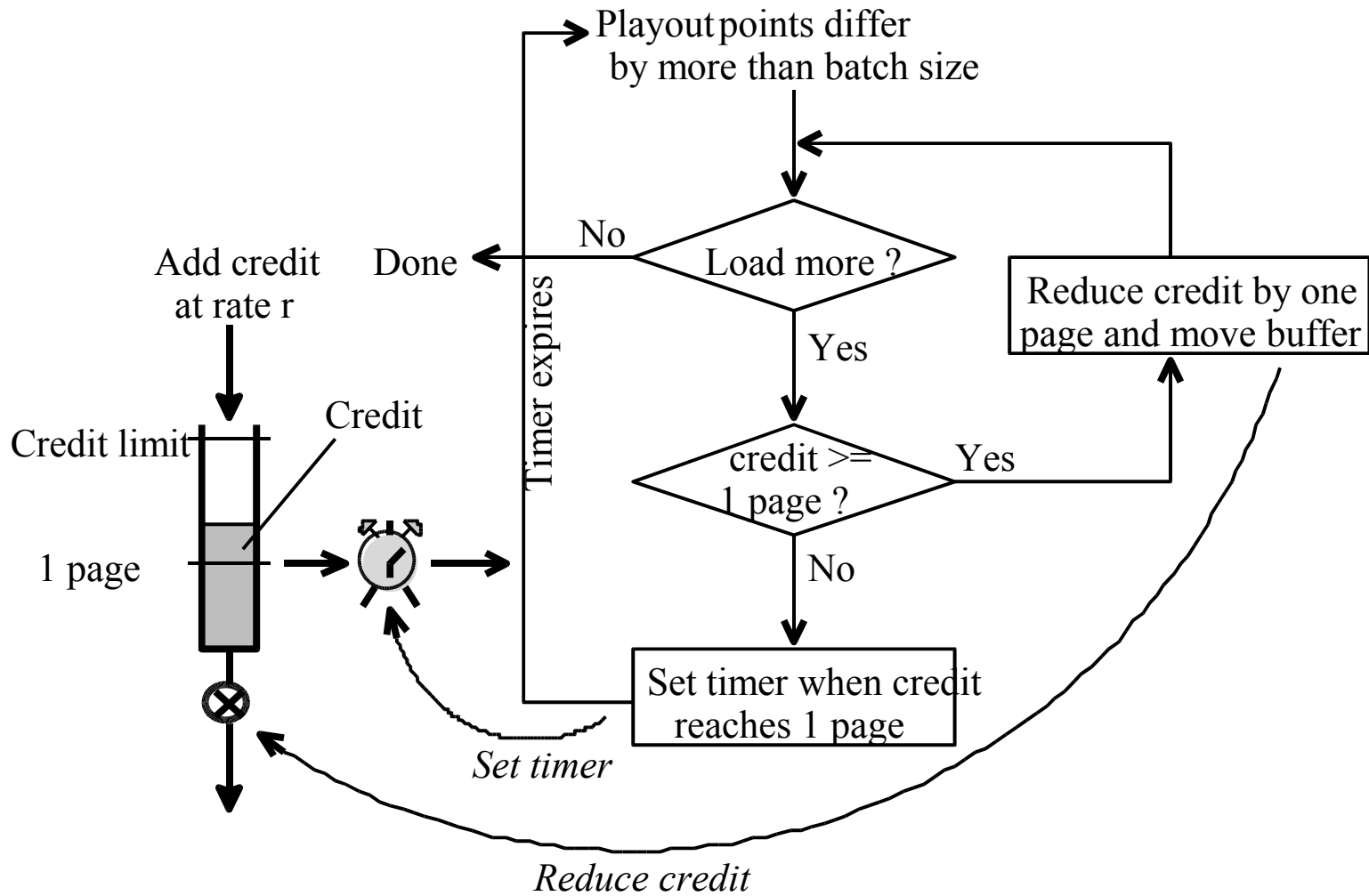
■ Page is accessible and up to date

■ Page is being loaded

↑ Pending read request

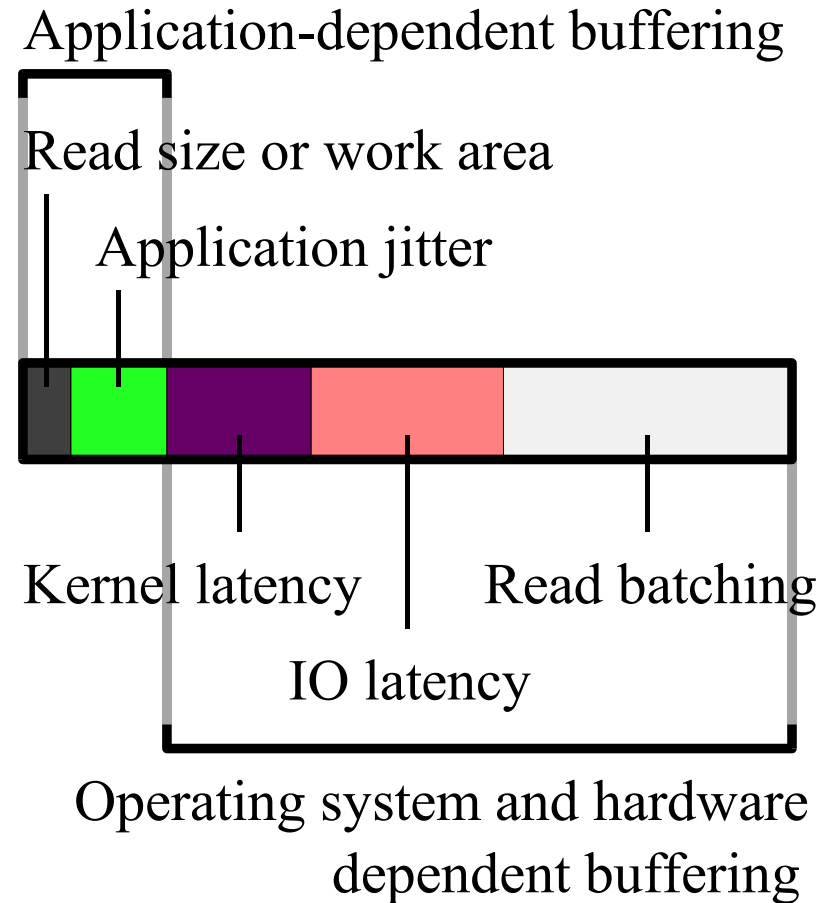


# Rate control



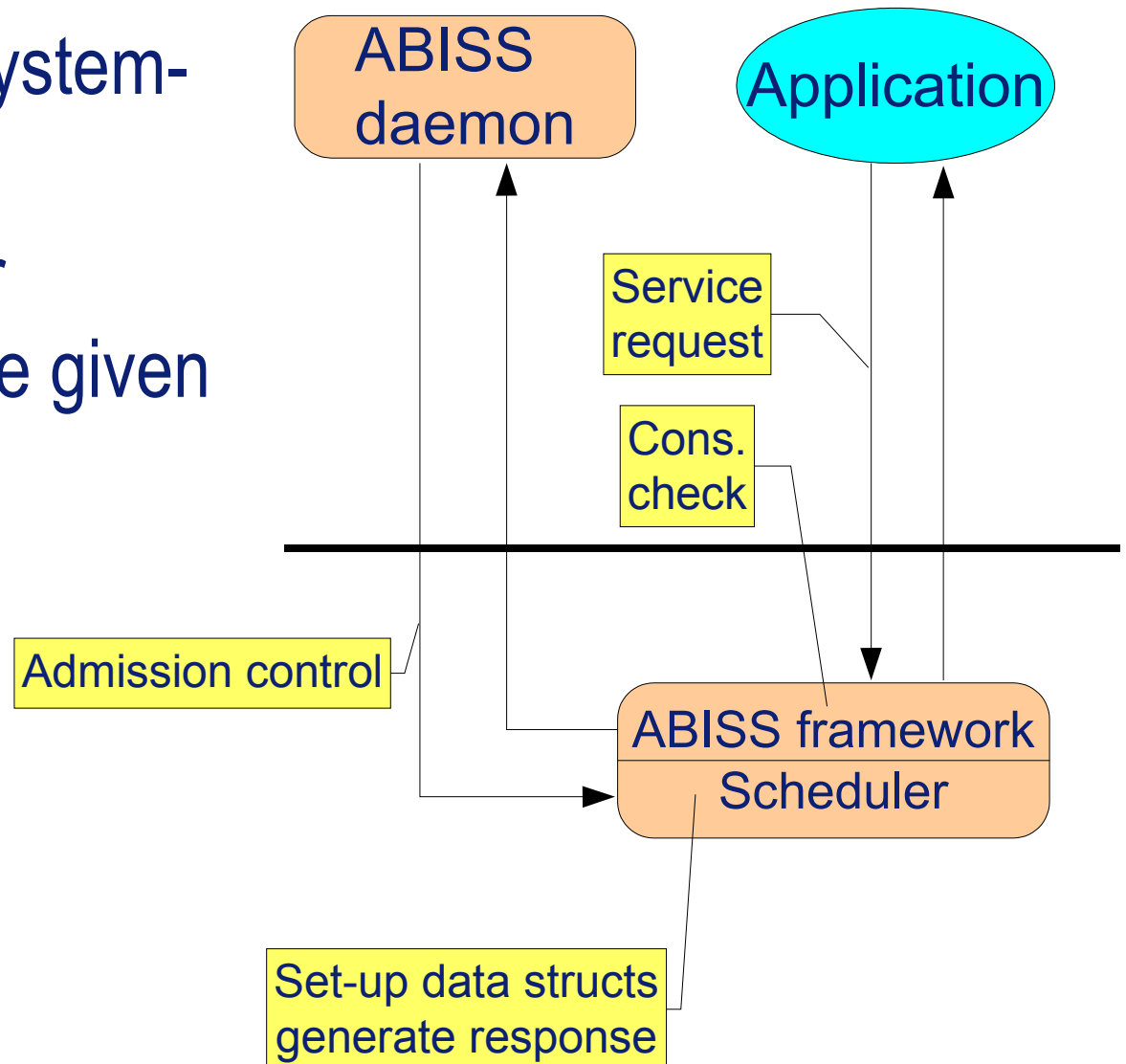
# Buffer size

- Application
  - Read size
  - 'jitter'
- Kernel
- Elevator + disk
- Read batching

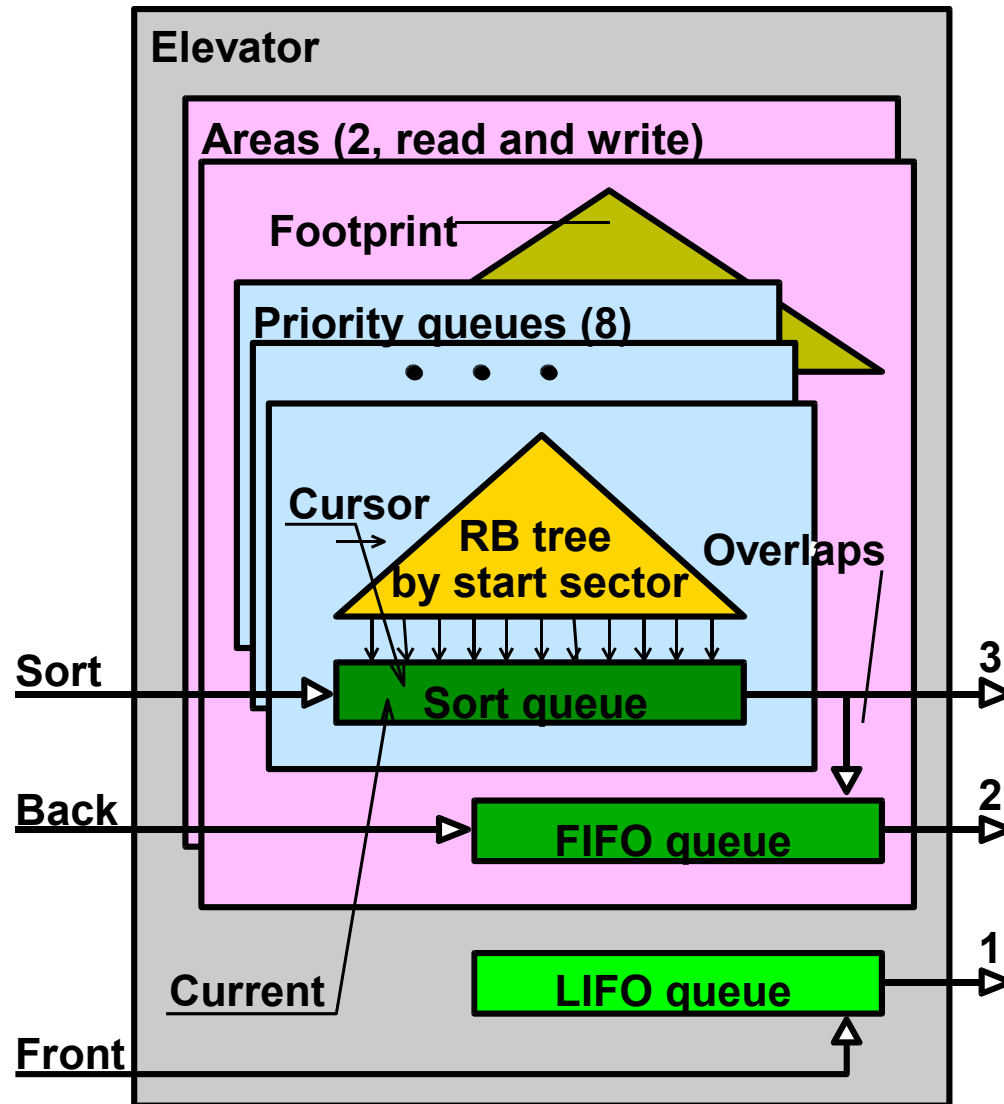


# User-space daemon

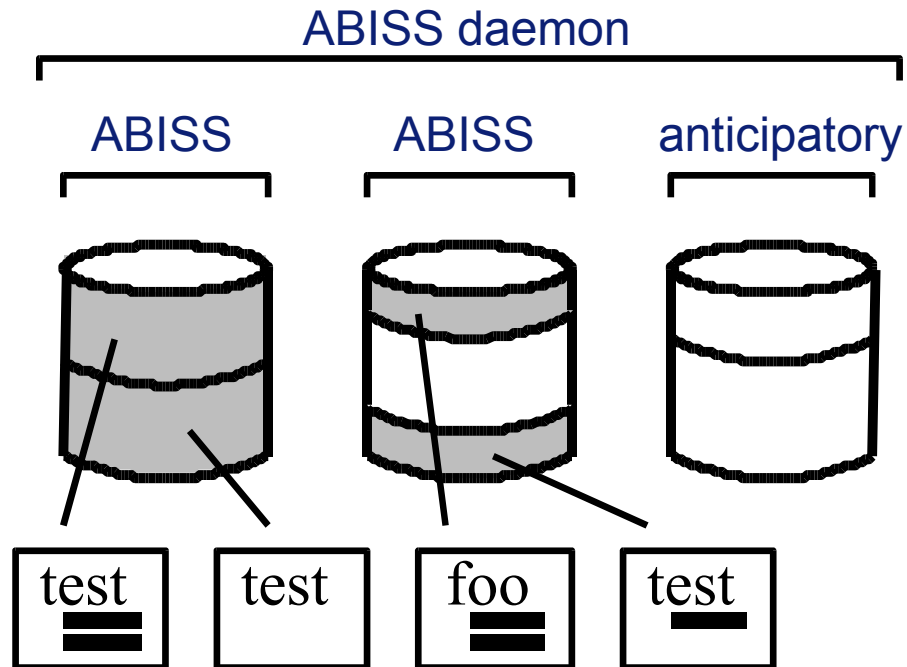
- Keeps track of system-wide use
- Decides whether bandwidth can be given
- e.g. Quotas



# elevator



# Scope of elevator and scheduler



# API: request ABISS service on file

```
static struct abiss_attach_msg msg;  
static struct abiss_sched_test_prm prm;
```

```
fd = open("name", O_RDONLY);  
msg.header.type = abiss_attach;  
prm.ra_bytes = app_buffer;  
prm.fill_Bps = byterate;  
msg.sched_prm = &prm;  
...  
if (ioctl(fd, ABISS_IOCTL, &msg) < 0)  
    /* handle error */;
```

```
FILE *abiss_fopen(const char *path, const char  
                 *mode, const int byterate, const int app_buffer)
```

# API: update playout point

```
static struct abiss_position_msg msg;
```

```
got = read(fd, buffer, BUFFER_SIZE);
```

```
msg.header.type = abiss_position;
```

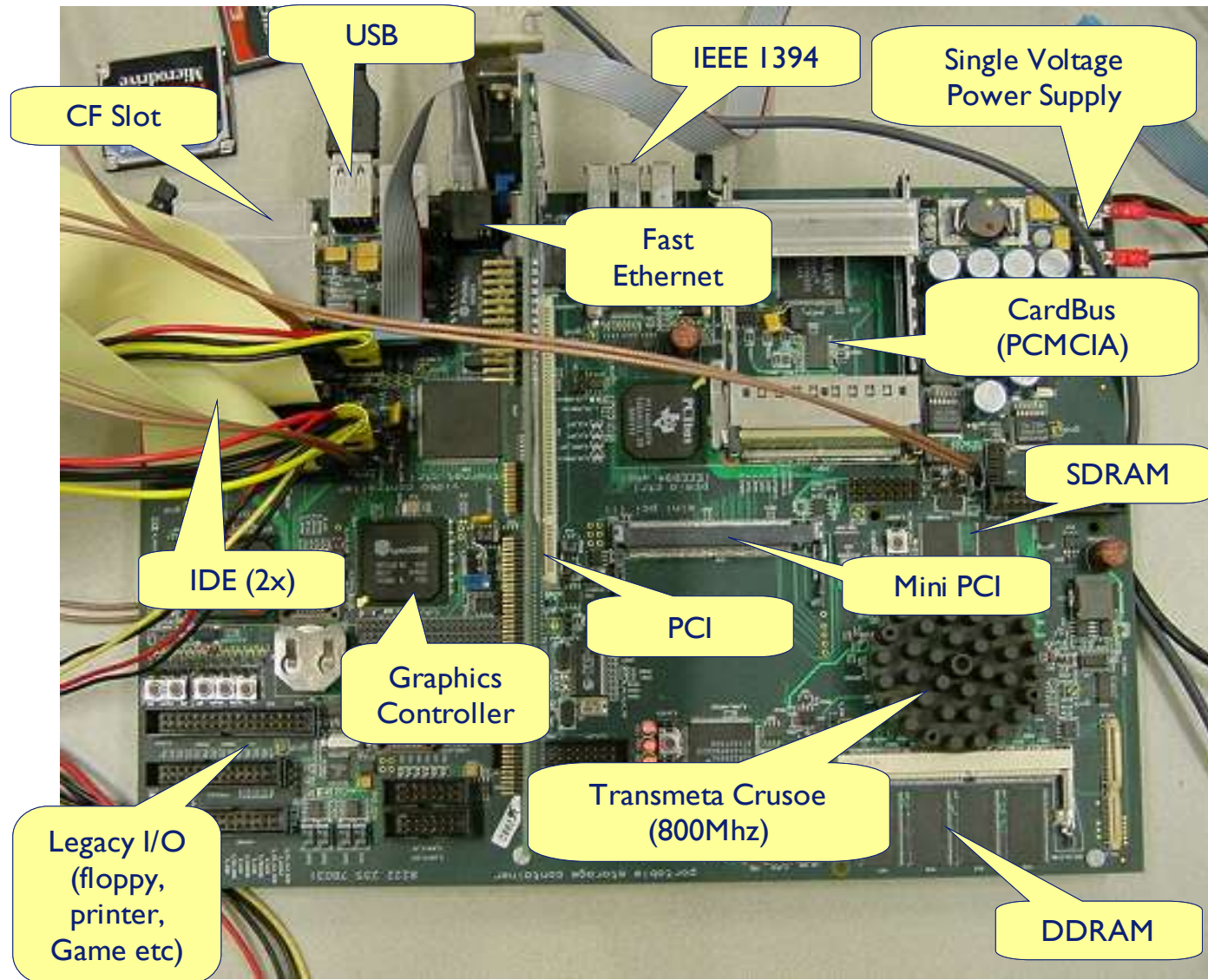
```
msg.pos = 0;
```

```
msg.whence = SEEK_CUR;
```

```
ioctl(fd, ABISS_IOCTL, &msg);
```

```
abiss_fread(void *ptr, size_t size, size_t nmemb,  
            FILE *fp)
```

# Measurement H/W setup



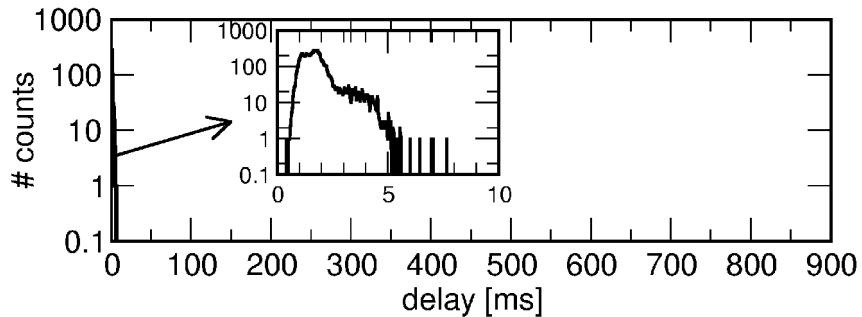


# Measurements, description

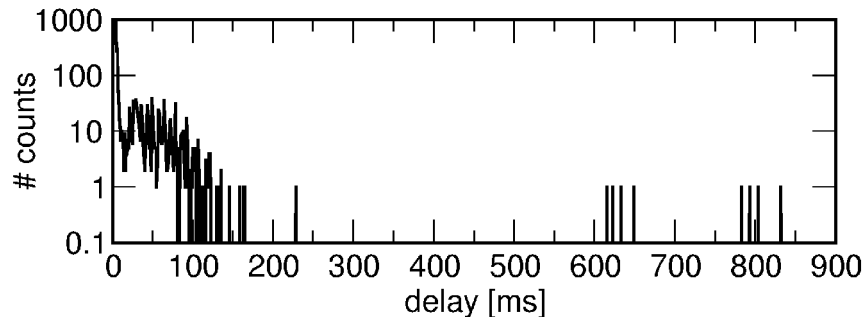
- Four RT streams;
  - 1 MB/s
  - 64 kB read size
  - 105 MB files
  - Playout buffer 564 kB
  - 20 pages batch size
- Measure time between read request and data retrieval
- Loop with BE file copy
  - 175 MB file
  - Read size 128 kB
- Different elevators:
  - ABISS RT
  - ABISS BE
  - Anticipatory
  - Deadline
  - CFQ
  - Noop

# Measurements, graphs

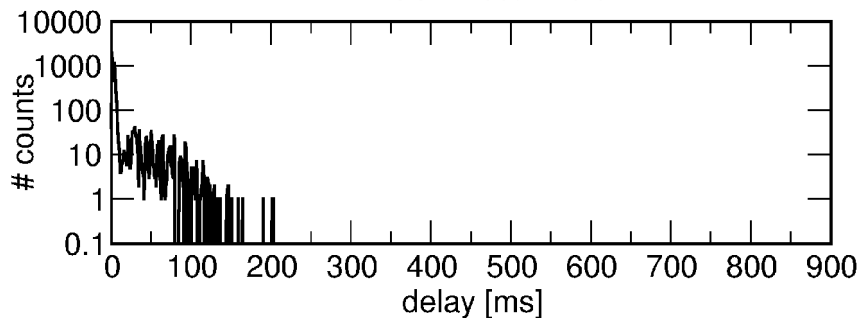
ABISS - Real Time



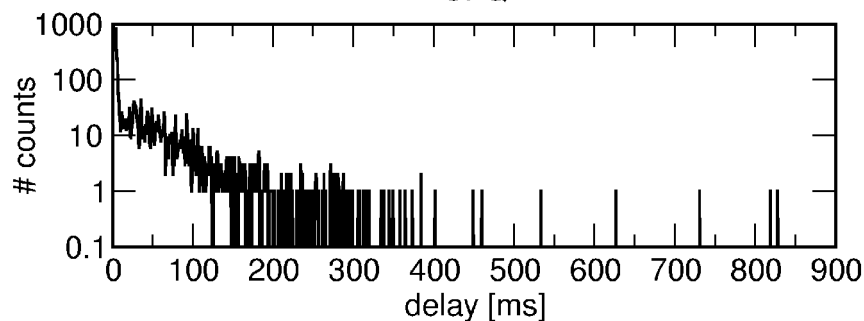
Deadline



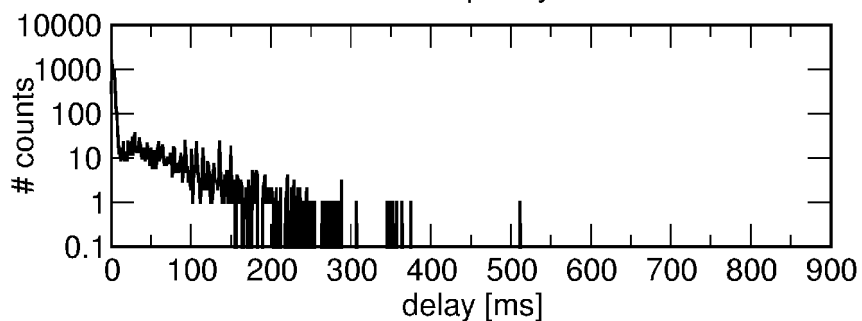
ABISS - Best Effort



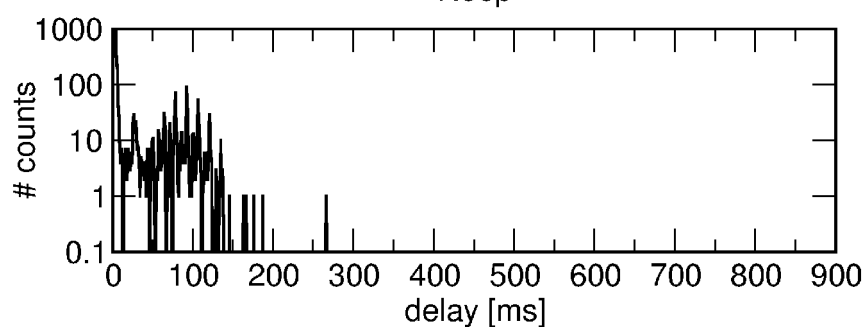
CFQ



Anticipatory



Noop



# BE performance

Elevator		BE reader performance [Mbytes/s]	
		One RT reader	Four RT readers
ABISS	RT, 10 pg batch	7.7	0.3
	RT, 20 pg batch	8.0	2.5
	RT, 40 pg batch	8.7	4.0
	RT, 80 pg batch	9.4	5.8
	RT, 160 pg batch	9.5	6.6
	BE	7.7	1.5
Anticipatory		7.8	2.7
Deadline		7.9	1.8
CFQ		7.9	1.8
Noop		7.9	2.0

# Future work

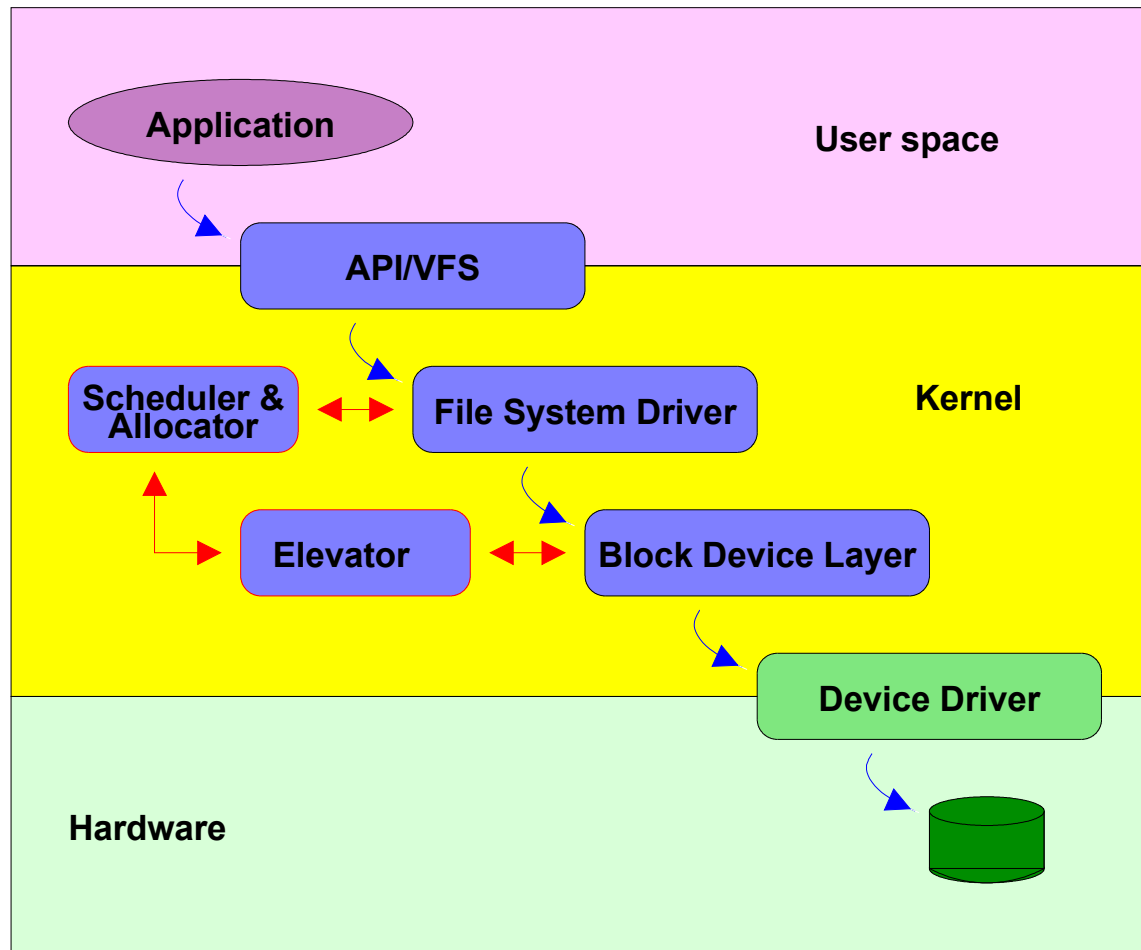
- Ext3 filesystem
- Add RT writing
  - Investigate buffering requirements
  - Allocator
- Different schedulers, e.g. Power for portable players
- Investigate Asynchronous I/O with elevator priorities

# Conclusions

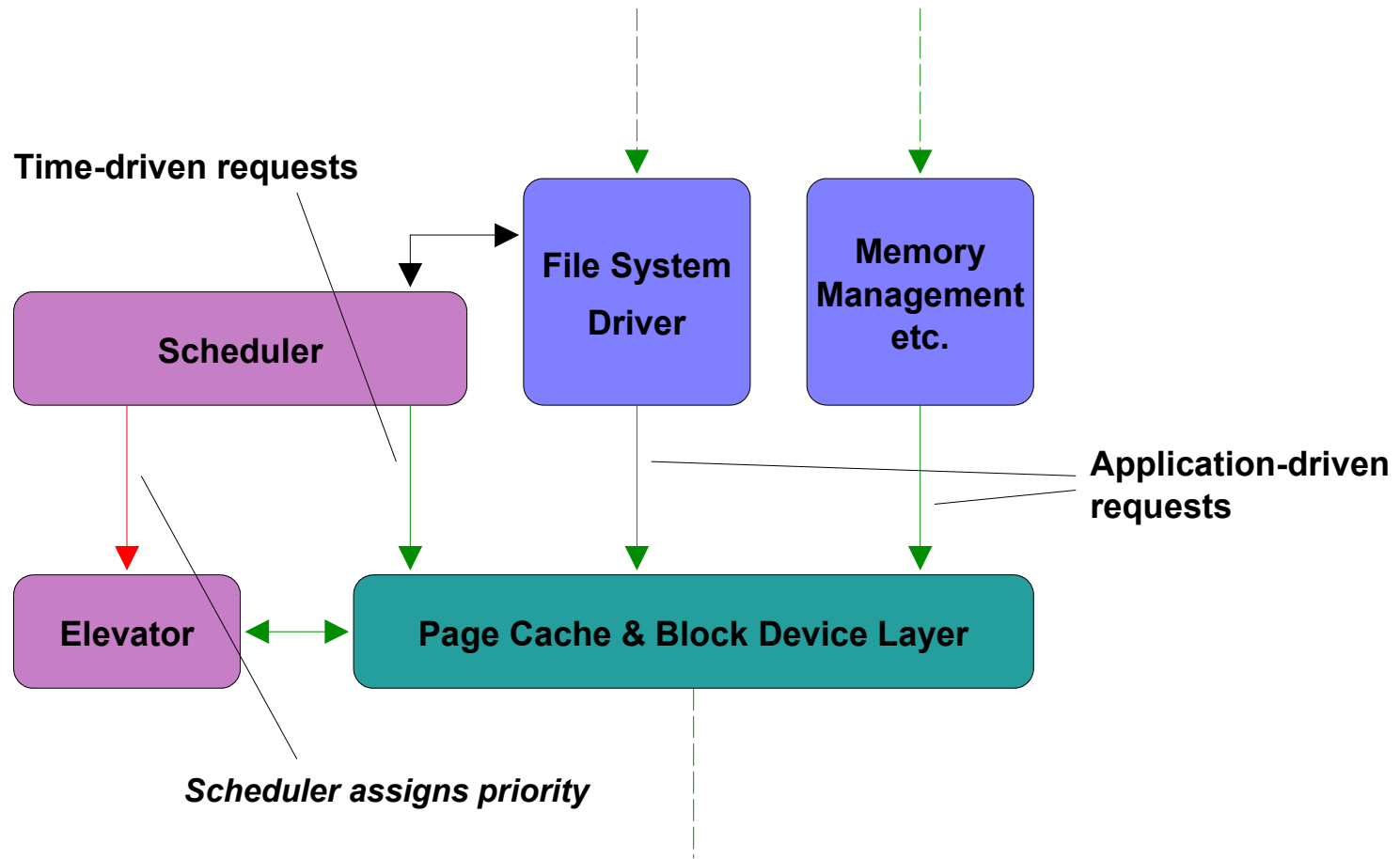
- ABISS framework
  - Allows for different, run-time switchable services
  - 'test' scheduler implemented
  - User-space daemon
- Evaluated performance 'test' scheduler
  - Guarantee bandwidth without need for large app. buffers
  - Favorable comparison to existing elevators
- <http://abiss.sourceforge.net/>

Questions?

# ABISS

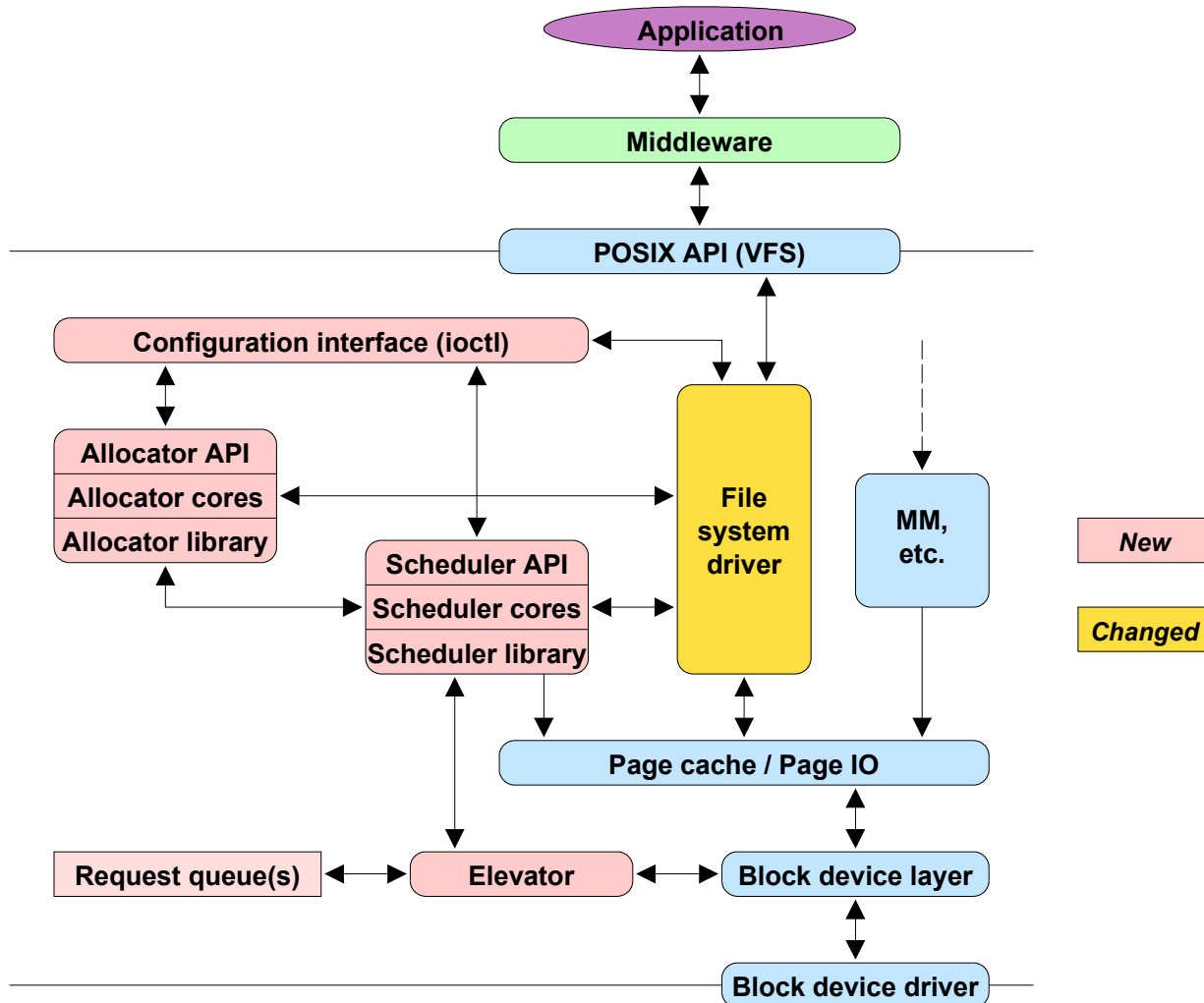


# scheduler

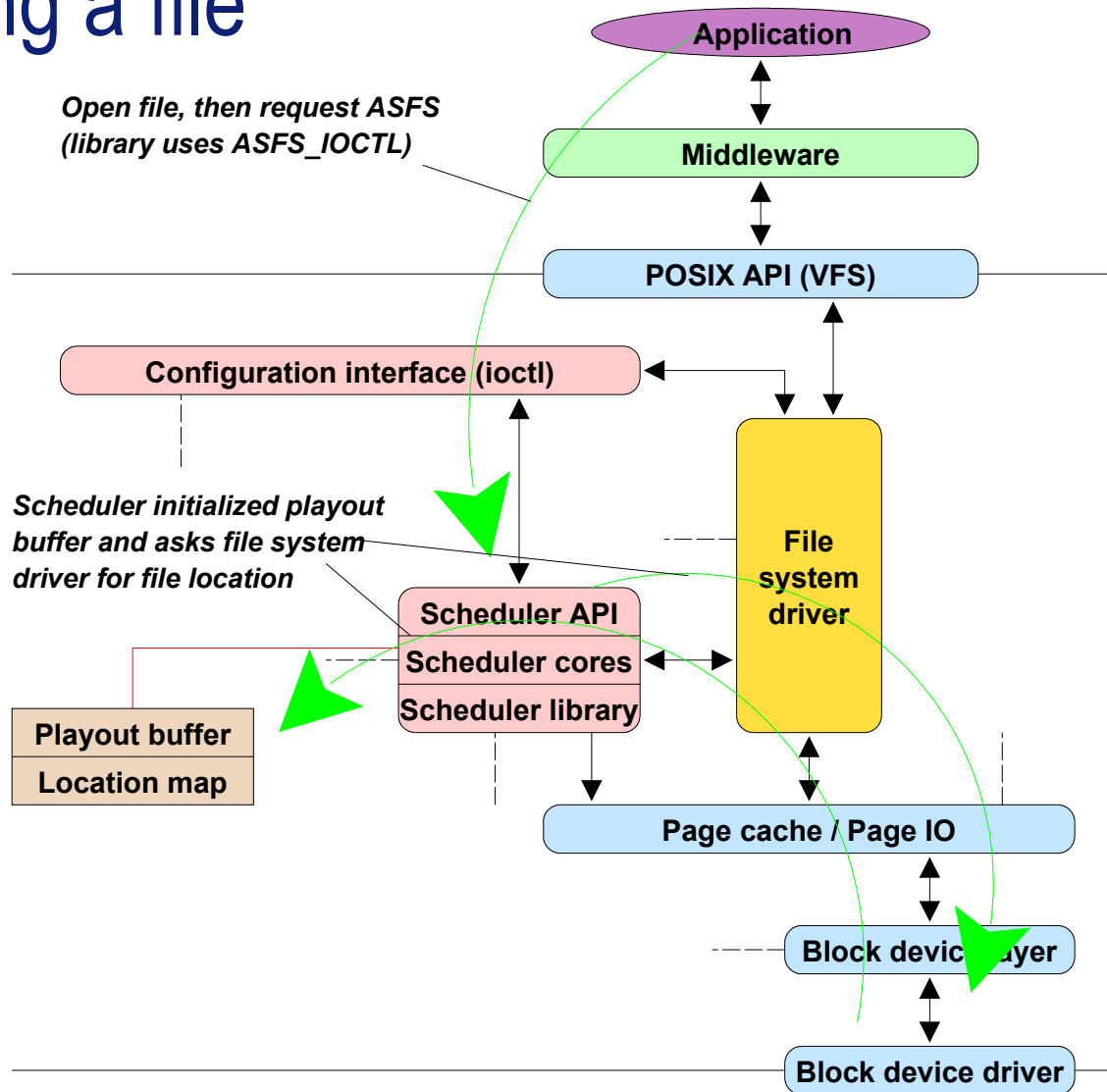




# system overview (detailed)

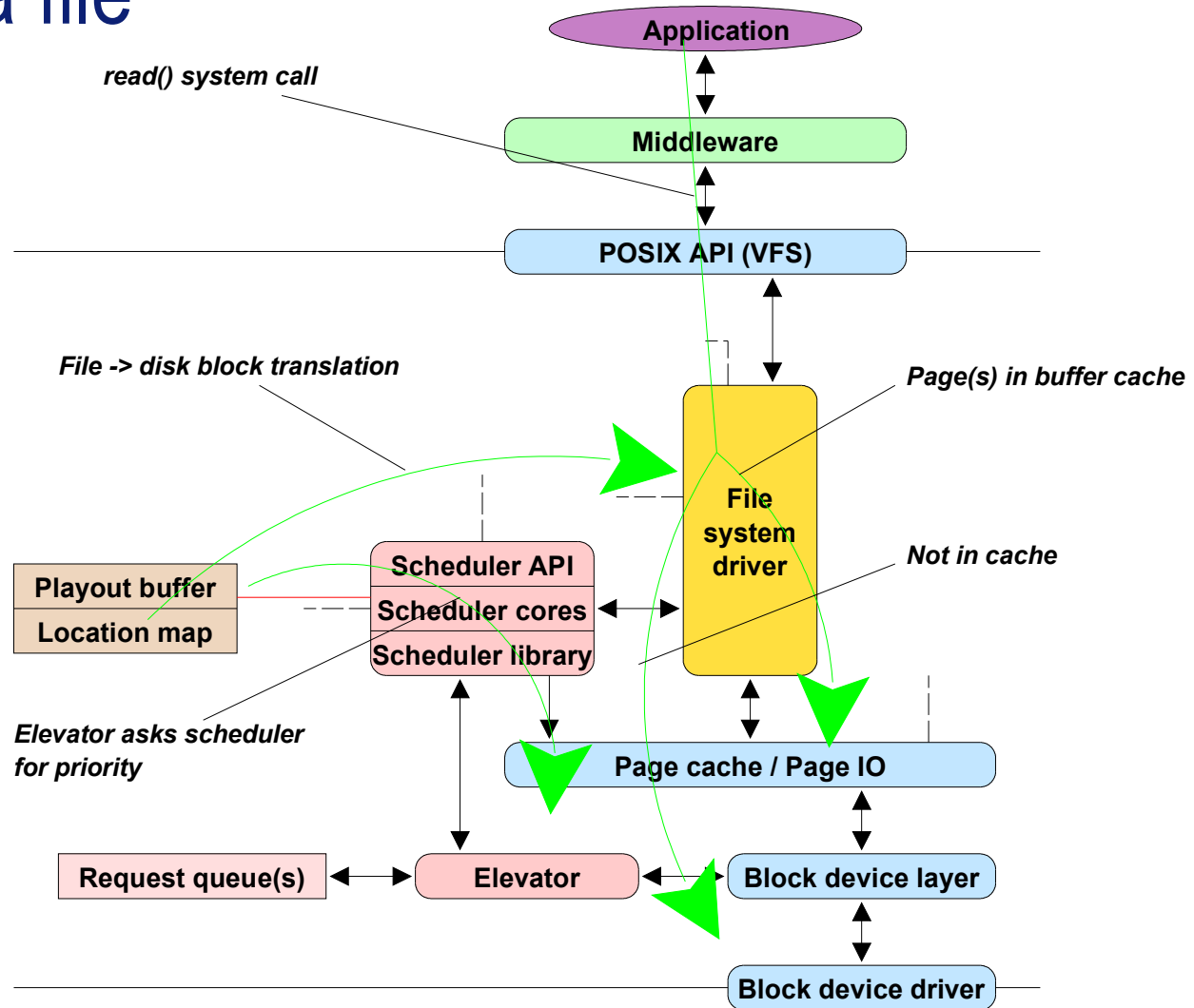


# opening a file

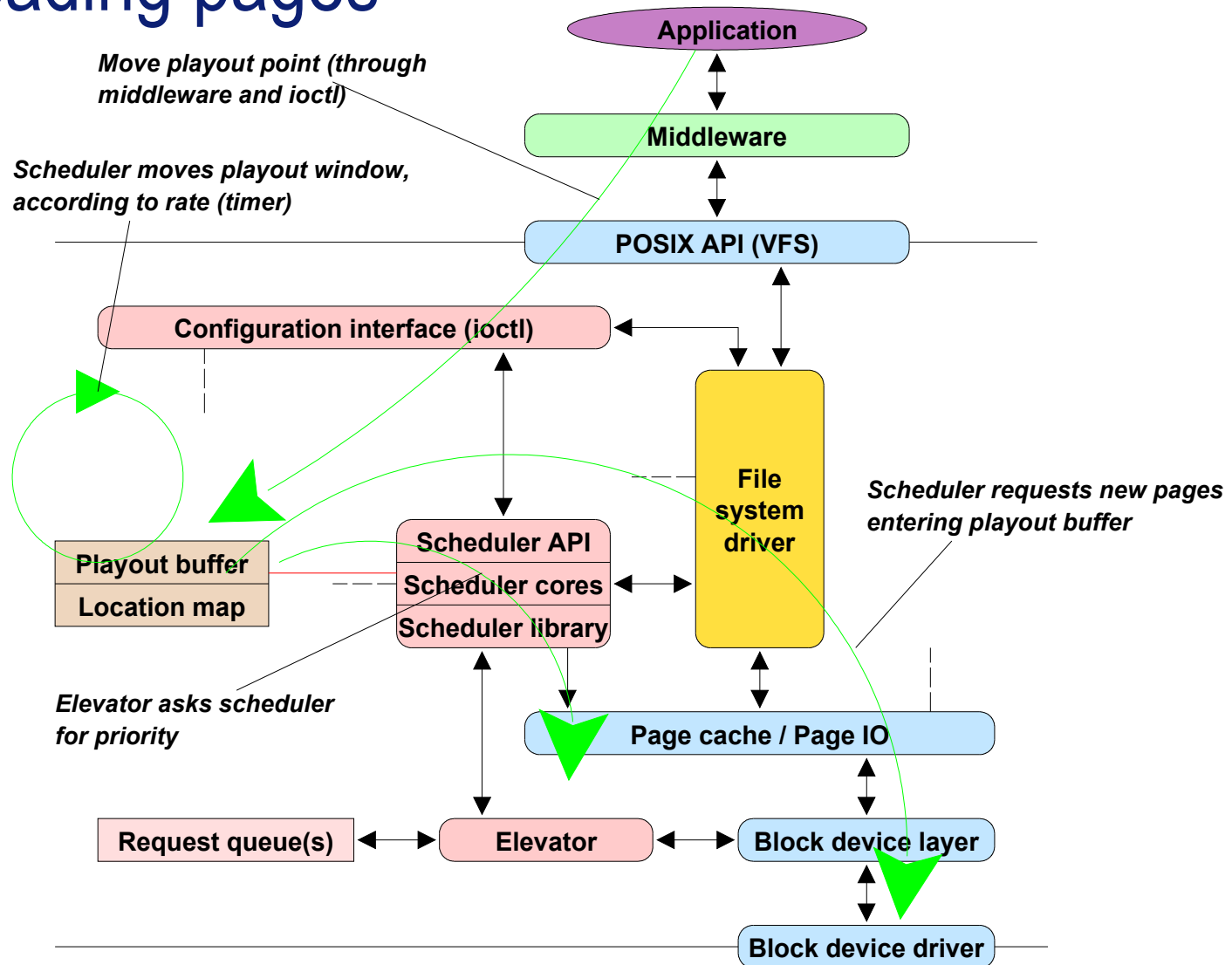


# reading a file

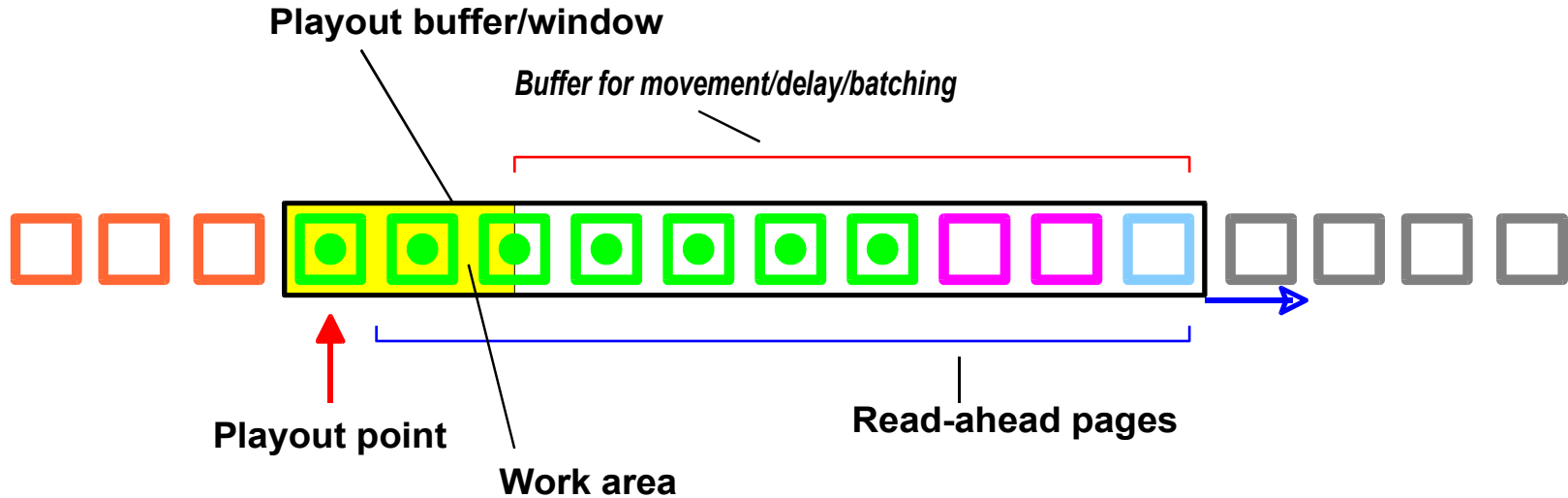
reading a file (read)







# preloading pages

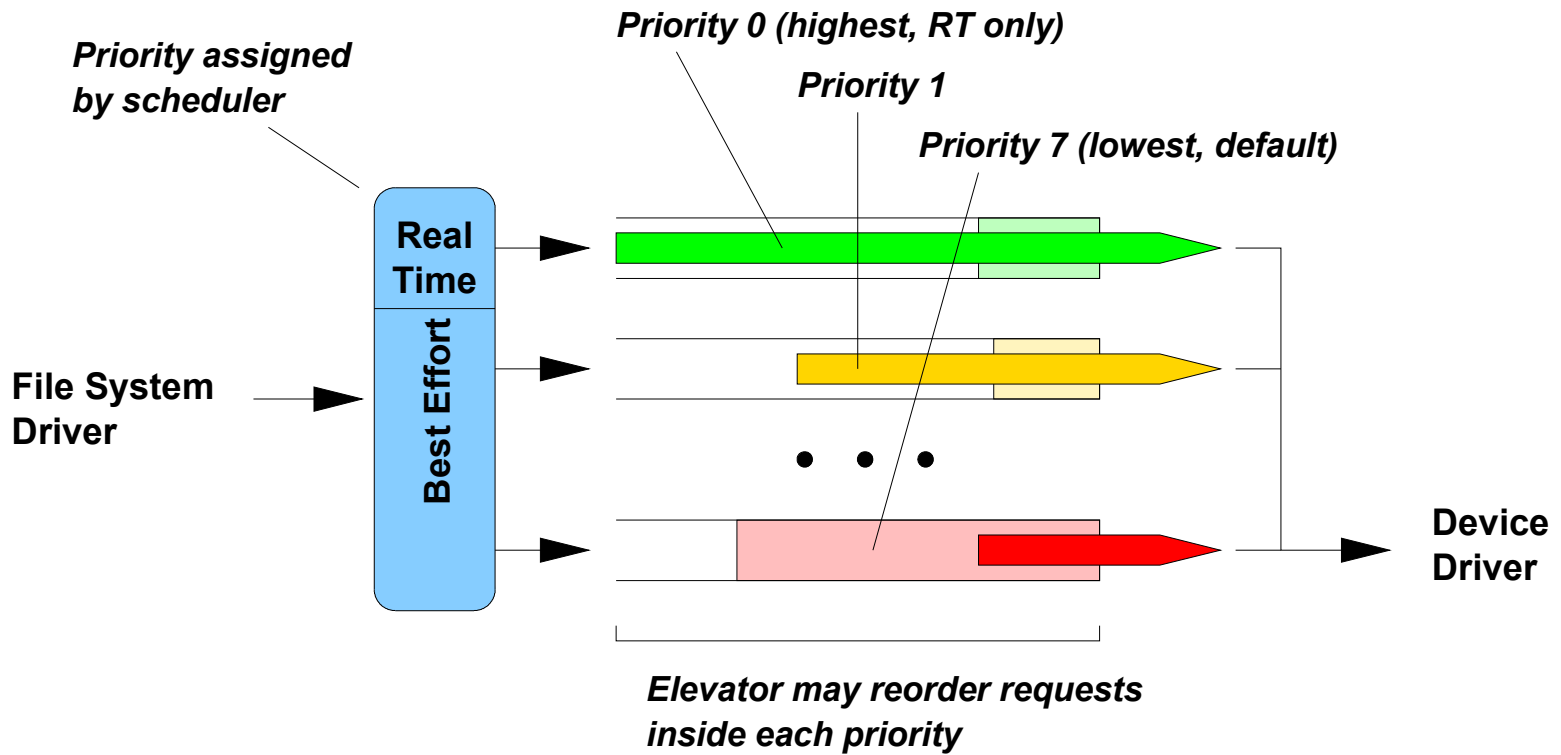


# playout buffer



-  *Page is loaded and locked in memory*
-  *Page is being loaded*
-  *Page will be loaded later*
-  *Page not used anymore*

# elevator



# Credit & moving playout point

