



Linux-Automation Konferenz 2005

# Anwendung und Erweiterung des Real-Time Driver Model

Hans-Peter Bock

<Hans-Peter.Bock@isw.uni-stuttgart.de>





- RTDM
  - Überblick
  - Schnittstelle zum Treiber
  - Schnittstelle zur Applikation
- select()
  - aktueller Stand
  - korrekte Lösung
- Zusammenfassung und Ausblick



# RTDM

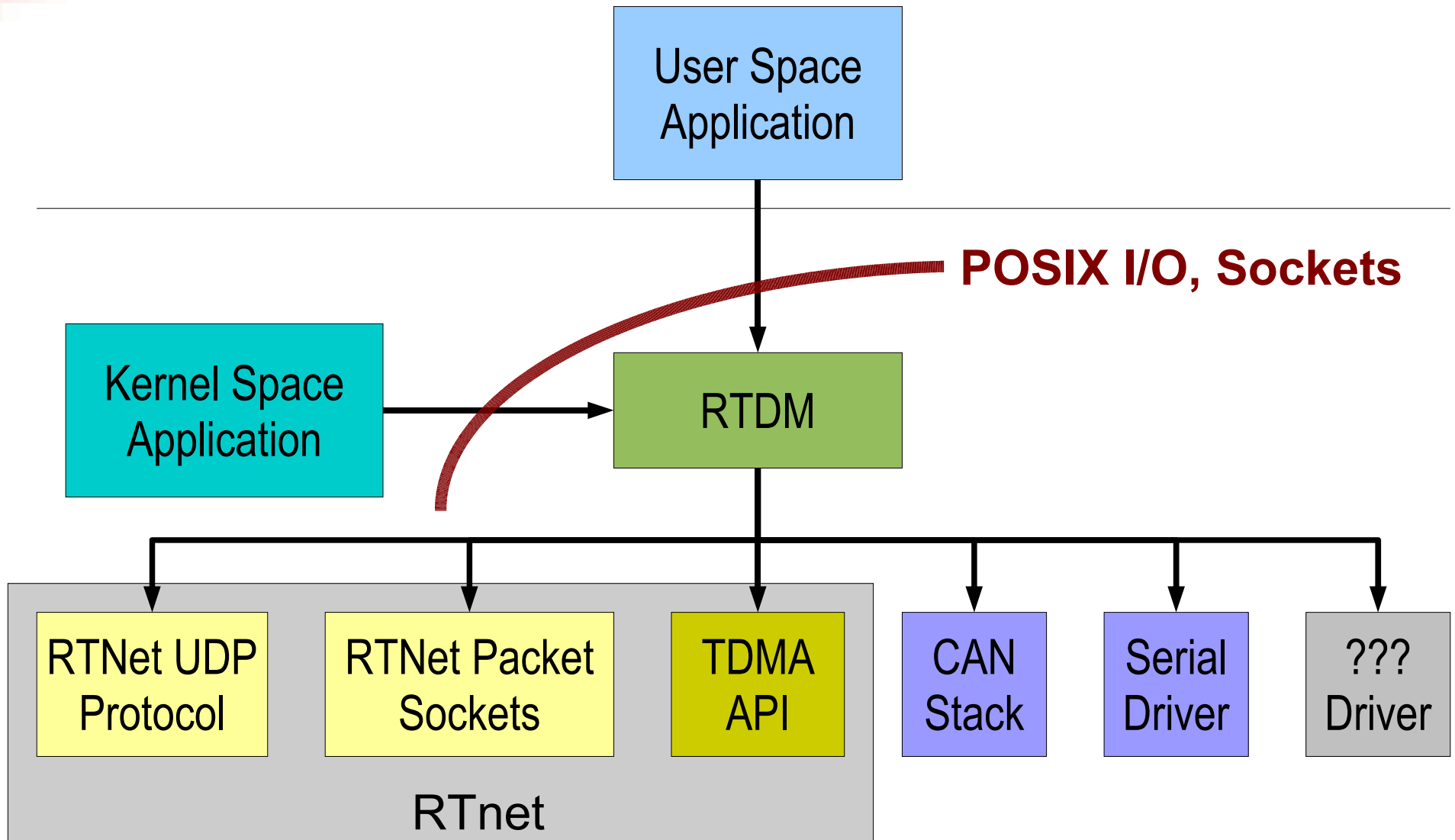
# Motivation für RTDM



- Autoren
  - Jörg Langenberg
  - Jan Kiszka
  
- Motivation für RTDM:
  - einheitliche Treiberschnittstelle für Applikationen
  - notwendig für eventgesteuerte Programmierung



# Real-Time Driver Model





- Treiber (de)registriert sich bei RTDM mit:
  - `rtm_dev_register(struct rtdm_device* dev);`
  - `rtm_dev_unregister(struct rtdm_device* dev);`
- Treiber stellt bestimmte Funktionen zur Verfügung:
  - `struct rtdm_device;`
  - `struct rtdm_operations;` } `rtdm_driver.h`

# struct rtdm\_device



```
struct rtdm_device {
    int                struct_version;

    int                device_flags;
    size_t             context_size;

    /* named device identification */
    char                device_name[RTDM_MAX_DEVNAME_LEN+1];

    /* protocol device identification */
    int                protocol_family;
    int                socket_type;

    /* device instance creation */
    open_handler       open_rt;
    open_handler       open_nrt;

    socket_handler     socket_rt;
    socket_handler     socket_nrt;

    struct rtdm_operations ops;

    int                device_class;
    int                device_sub_class;
    const char         *driver_name;
    const char         *peripheral_name;
    const char         *provider_name;

    /* /proc entry */
    const char         *proc_name;
    struct proc_dir_entry *proc_entry;

    /* driver-definable id */
    int                device_id;

    struct rtdm_dev_reserved reserved;
};
```

open\_handler open\_rt;  
open\_handler open\_nrt;



# struct rtdm\_device



```
struct rtdm_device {
    int                struct_version;

    int                device_flags;
    size_t             context_size;

    /* named device identification */
    char               device_name[RTDM_MAX_DEVNAME_LEN+1];

    /* protocol device identification */
    int                protocol_family;
    int                socket_type;

    /* device instance creation */
    open_handler       open_rt;
    open_handler       open_nrt;

    socket_handler     socket_rt;
    socket_handler     socket_nrt;

    struct rtdm_operations ops;

    int                device_class;
    int                device_sub_class;
    const char         *driver_name;
    const char         *peripheral_name;
    const char         *provider_name;

    /* /proc entry */
    const char         *proc_name;
    struct proc_dir_entry *proc_entry;

    /* driver-definable id */
    int                device_id;

    struct rtdm_dev_reserved reserved;
};
```

socket\_handler socket\_rt;  
socket\_handler socket\_nrt;



# struct rtdm\_device



```
struct rtdm_device {
    int                struct_version;

    int                device_flags;
    size_t             context_size;

    /* named device identification */
    char                device_name[RTDM_MAX_DEVNAME_LEN+1];

    /* protocol device identification */
    int                protocol_family;
    int                socket_type;

    /* device instance creation */
    open_handler       open_rt;
    open_handler       open_nrt;

    socket_handler     socket_rt;
    socket_handler     socket_nrt;

    struct rtdm_operations ops;

    int                device_class;
    int                device_sub_class;
    const char         *driver_name;
    const char         *peripheral_name;
    const char         *provider_name;

    /* /proc entry */
    const char         *proc_name;
    struct proc_dir_entry *proc_entry;

    /* driver-definable id */
    int                device_id;

    struct rtdm_dev_reserved reserved;
};
```

struct rtdm\_operations ops;



# struct rtdm\_operations



```
struct rtdm_operations {
```

```
/* common operations */  
close_handler      close_rt;  
close_handler      close_nrt;  
ioctl_handler      ioctl_rt;  
ioctl_handler      ioctl_nrt;
```

```
/* stream-oriented device operations */  
read_handler       read_rt;  
read_handler       read_nrt;  
write_handler      write_rt;  
write_handler      write_nrt;
```

```
/* message-oriented device operations */  
recvmsg_handler    recvmsg_rt;  
recvmsg_handler    recvmsg_nrt;  
sendmsg_handler    sendmsg_rt;  
sendmsg_handler    sendmsg_nrt;
```

```
[...]
```

```
};
```

```
close_handler close_rt;  
close_handler close_nrt;  
ioctl_handler ioctl_rt;  
ioctl_handler ioctl_nrt;
```



# struct rtdm\_operations



```
struct rtdm_operations {  
  
    /* common operations */  
    close_handler      close_rt;  
    close_handler      close_nrt;  
    ioctl_handler      ioctl_rt;  
    ioctl_handler      ioctl_nrt;  
  
    /* stream-oriented device operations */  
    read_handler       read_rt;  
    read_handler       read_nrt;  
    write_handler      write_rt;  
    write_handler      write_nrt;  
  
    /* message-oriented device operations */  
    recvmmsg_handler   recvmmsg_rt;  
    recvmmsg_handler   recvmmsg_nrt;  
    sendmsg_handler    sendmsg_rt;  
    sendmsg_handler    sendmsg_nrt;  
  
    [...]  
};
```


```
read_handler read_rt;  
read_handler read_nrt;  
write_handler write_rt;  
write_handler write_nrt;
```



# struct rtdm\_operations



```
struct rtdm_operations {  
  
    /* common operations */  
    close_handler      close_rt;  
    close_handler      close_nrt;  
    ioctl_handler      ioctl_rt;  
    ioctl_handler      ioctl_nrt;  
  
    /* stream-oriented device operations */  
    read_handler        read_rt;  
    read_handler        read_nrt;  
    write_handler       write_rt;  
    write_handler       write_nrt;  
  
    /* message-oriented device operations */  
    recvmmsg_handler    recvmmsg_rt;  
    recvmmsg_handler    recvmmsg_nrt;  
    sendmmsg_handler    sendmmsg_rt;  
    sendmmsg_handler    sendmmsg_nrt;  
  
    [...]  
};
```



```
recvmmsg_handler recvmmsg_rt;  
recvmmsg_handler recvmmsg_nrt;  
sendmmsg_handler sendmmsg_rt;  
sendmmsg_handler sendmmsg_nrt;
```





- POSIX I/O-Schnittstelle für *named devices*
  - `open_rt()`; `close_rt()`; `ioctl_rt()`;
  - `read_rt()`; `write_rt()`;
- Socket-API für *protocol devices*
  - `socket_rt()`; `close_rt()`; `ioctl_rt()`;
  - `recvmsg_rt()`; `sendmsg_rt()`;
- Andere Funktionen sind abgebildet auf
  - `recv/sendmsg` oder
  - IOCTLS
- Profile welche Funktionen und IOCTLS ein Treiber anbieten soll




# Implementierung von select()

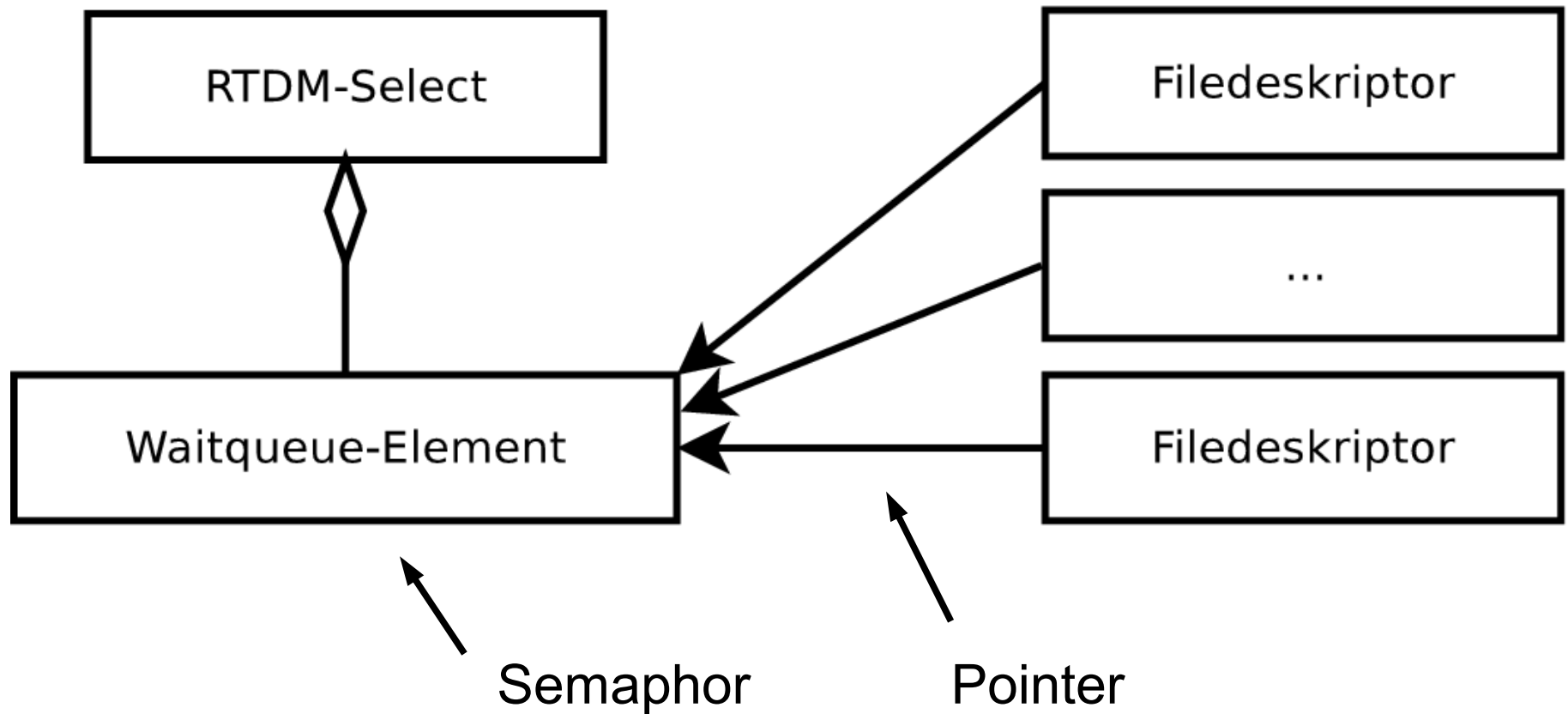


# Motivation für select()

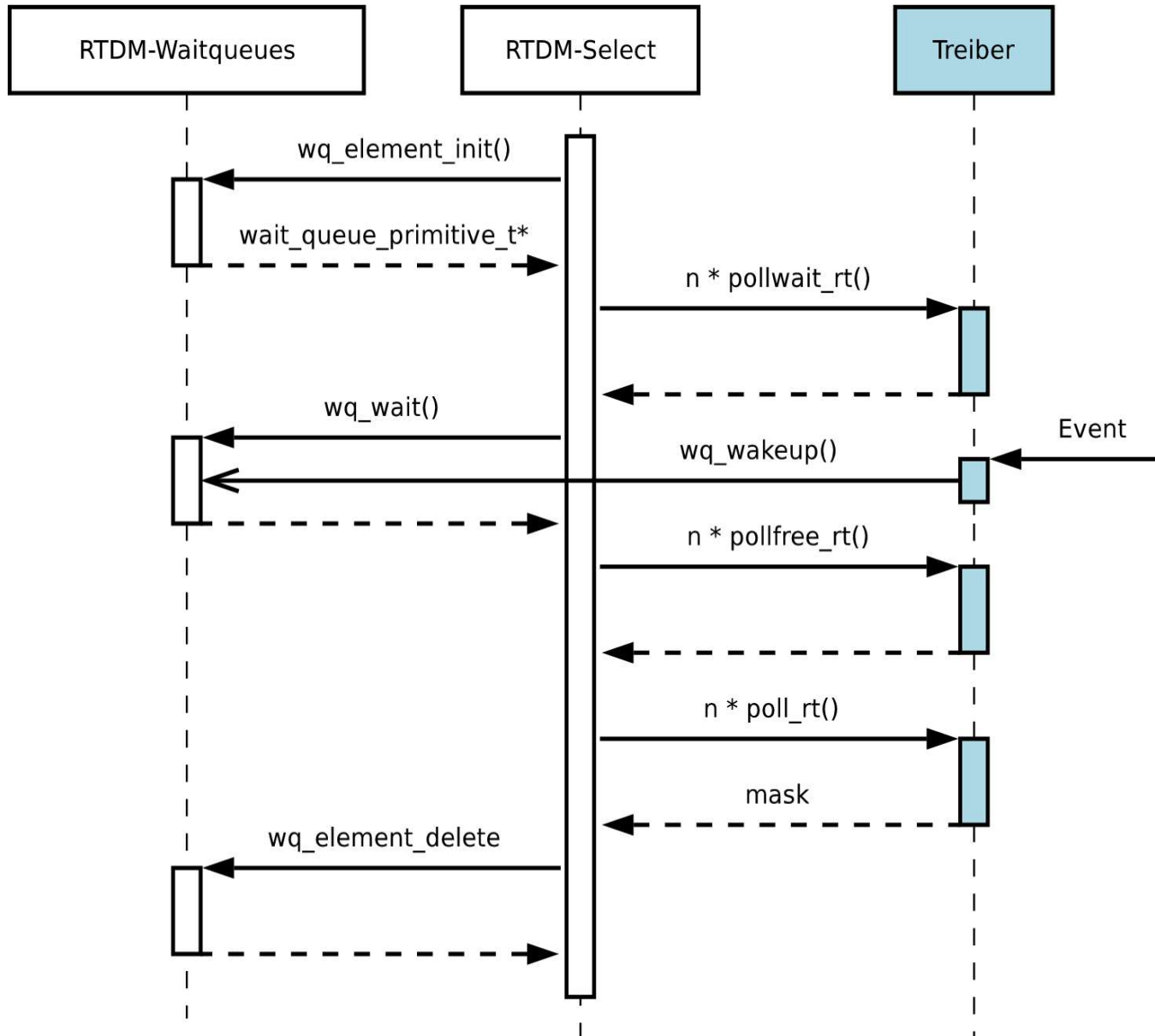


- eventgesteuerte Programmierung vermeidet
  - ressourcenintensives Polling
  - große Anzahl an Tasks
  - Beispiel: einfacher Webserver
  
- EU-Projekt  **CCEAN**  
OPEN CONTROLLER ENABLED BY AN  
ADVANCED REAL-TIME NETWORK
  - verteilte Steuerungsplattform mit ACE/TAO





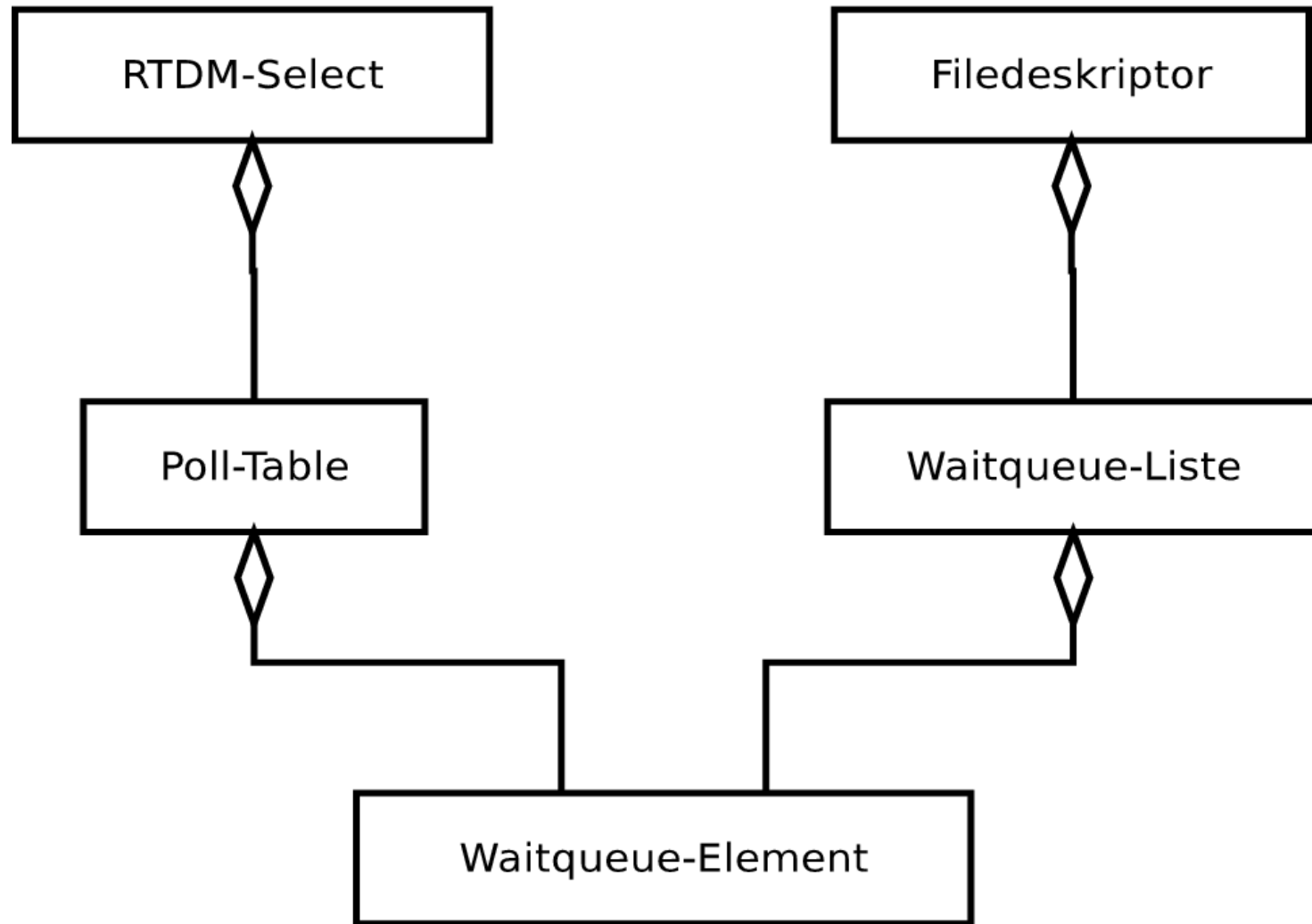
# Aktueller Ablauf



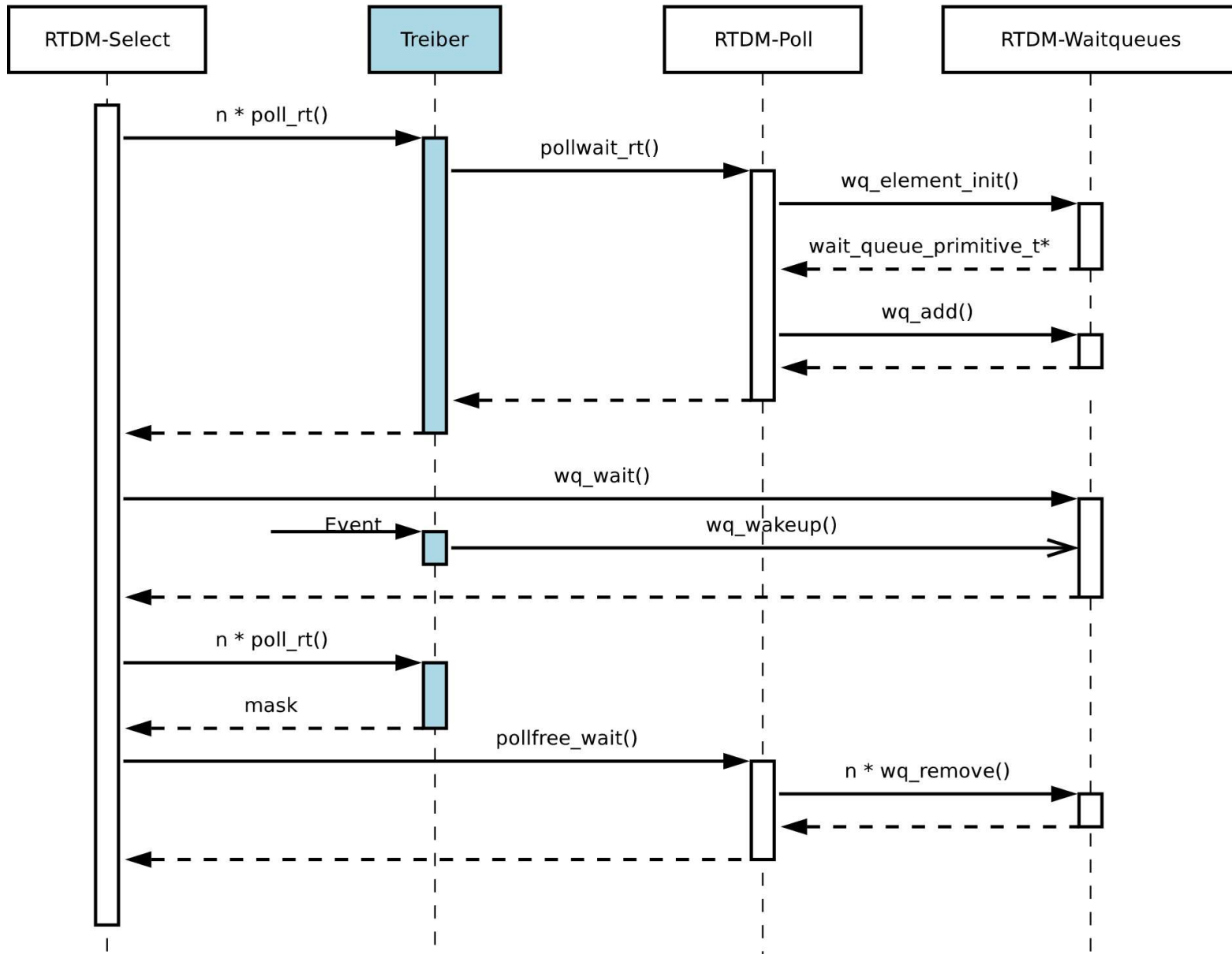


- Waitqueues sind noch keine *Queues*
  - maximal ein Task kann pro Filedeskriptor warten
- Funktionen im Treiber sollten ins RTDM:
  - `pollwait_rt()`
  - `pollfree_rt()`
- Timeout wird noch nicht berücksichtigt

# Korrekte Implementierung



# Korrektter Ablauf



# Roadmap für vollständigeres select()



- Waitqueues
  - dynamisch oder
  - mit Pool für Elemente
- pollwait\_rt() nach RTDM
- pollfree\_rt() nach RTDM
- Timeout einprogrammieren





# Zusammenfassung und Ausblick



## Zusammenfassung

- RTDM vorgestellt
- Proof-of-concept für `select()` existiert bereits
- mit Waitqueues kann `select()` erweitert werden

## Ausblick

- Einbau von RTDM in Fusion



Vielen Dank für Ihre Aufmerksamkeit!

Hans-Peter Bock  
<Hans-Peter.Bock@isw.uni-stuttgart.de>

<http://www.isw.uni-stuttgart.de/>