FIRST
Flexible Integrated Real-Time Systems Technology

ist
information society technologies

FIRST FRAMEWORK ON SHaRK OS

SCUOLA SUPERIORE S. ANNA

Giuseppe Lipari, Michael Trimarchi

RETIS Lab

Scuola Superiore Sant'Anna

Summary

Software Framework

Status of implementation in Shark

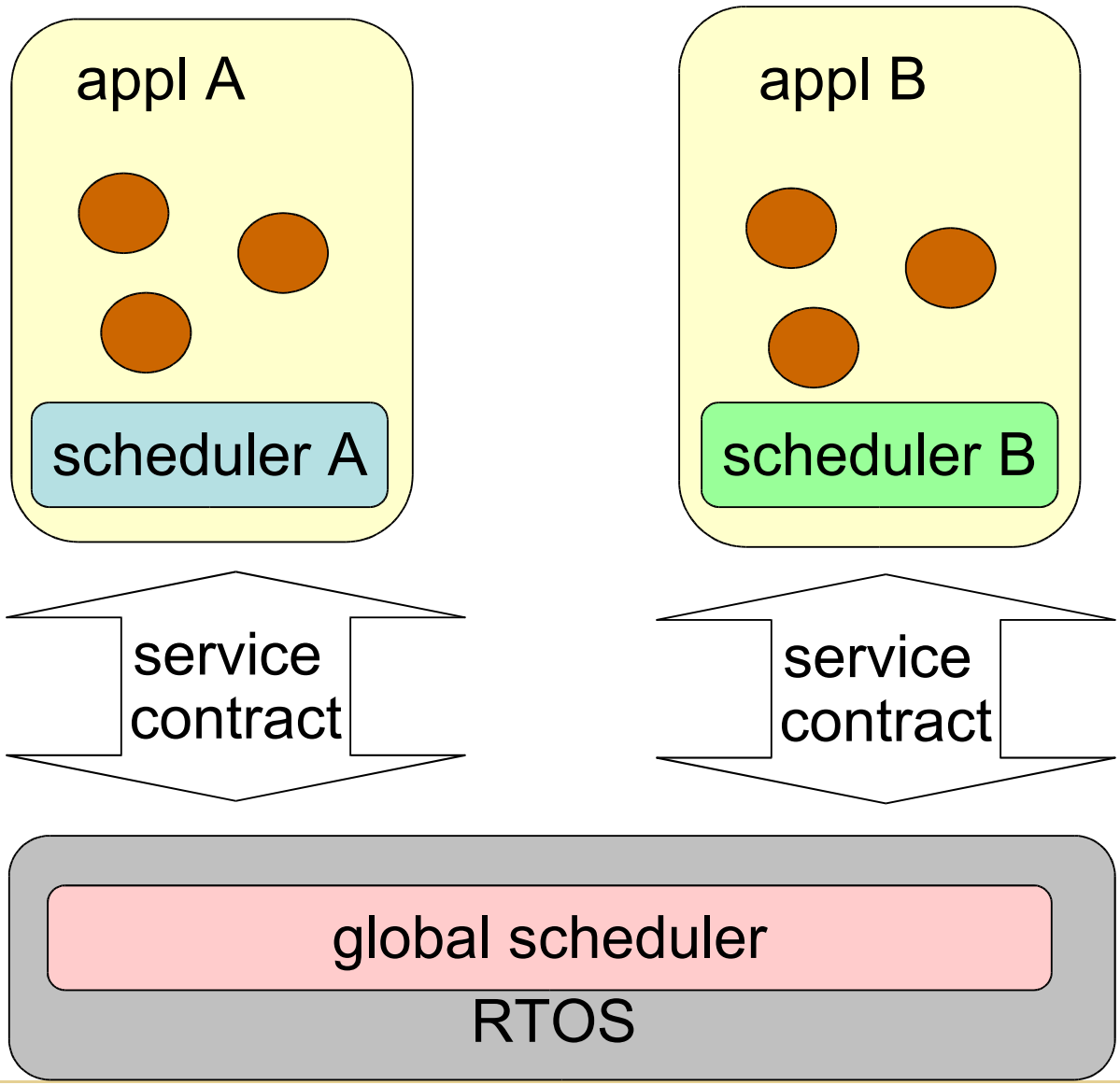
Examples of usage of the API

Creating a contract for

- periodic, sporadic hard real-time tasks
- soft real-time task
- imprecise computation
- applications (set of tasks)

Software Framework

- **Application = set of tasks/threads (+ scheduler)**
  - Synonyms: Component, Subsystem
  - It can be reduced to one single task
- **Hierarchical scheduling structure**
  - System = set of applications
  - Each application may have its own local scheduler
- **Service Contract**
  - Each application specifies its requirements by requiring a *service contract*

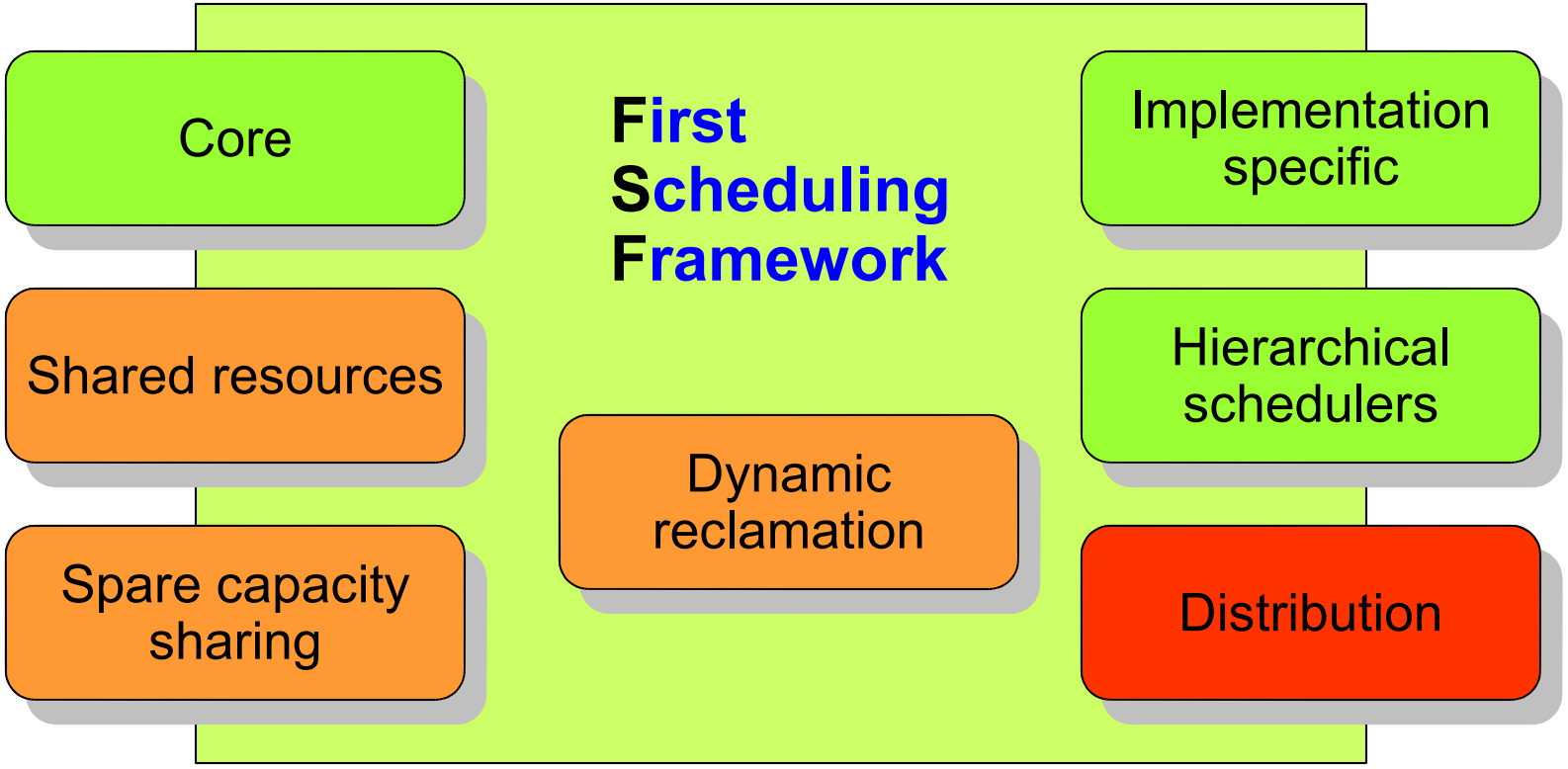

appl A
scheduler A
appl B
scheduler B
service contract
service contract
global scheduler
RTOS

Server based scheduling

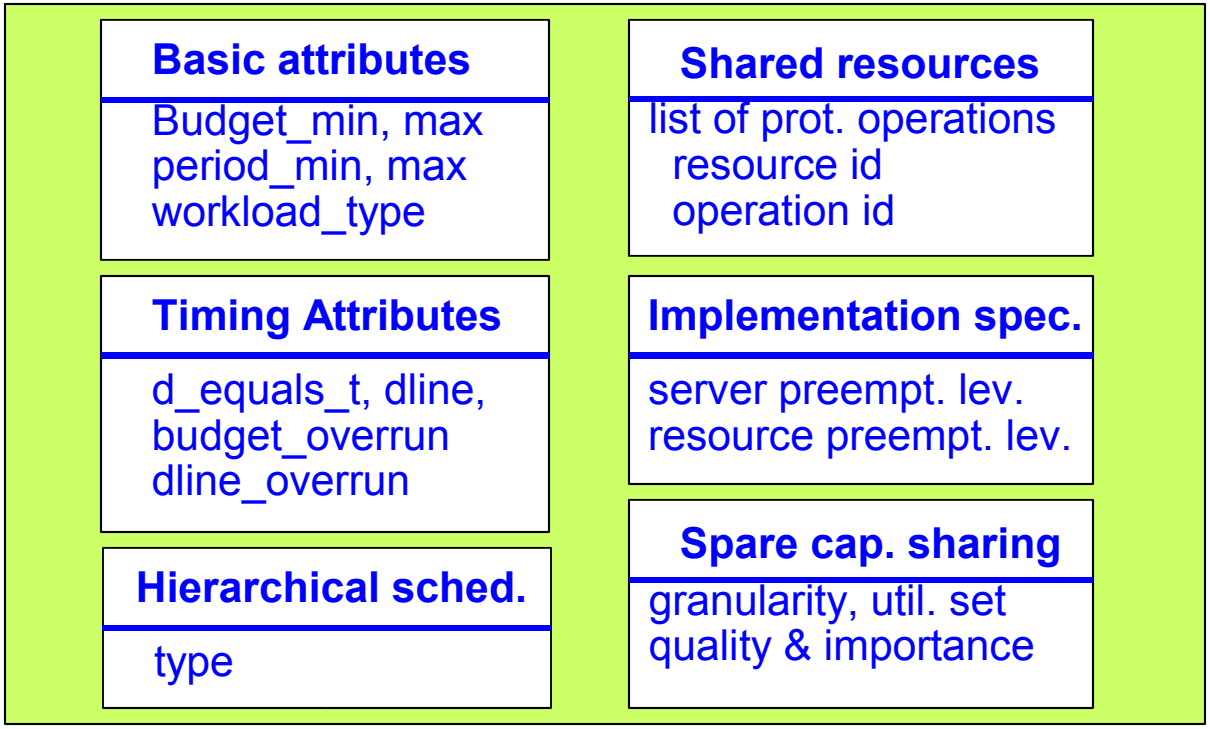
- **Server-based scheduling**
  - Each application is assigned *one or more* servers
  - Each server has a budget and a period
  - Provides temporal isolation
  - Provides independent analysis

- **Server algorithm**
  - No specific global scheduling strategy
  - No specific server algorithm
  - Systems can be based on
    - Fixed Priority and Sporadic Server
    - EDF and Constant Bandwidth Server
    - Table Driven and Slot Shifting

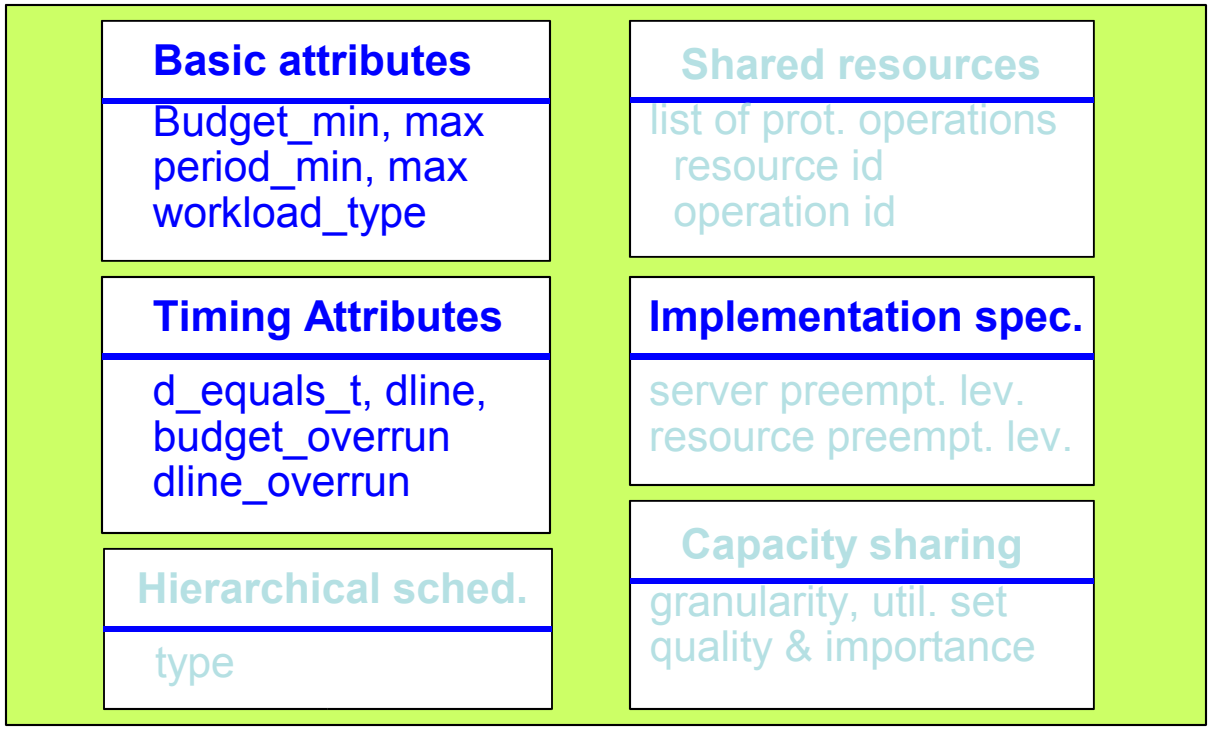
Status of implementation in Shark



Service contract parameters



Core service contract

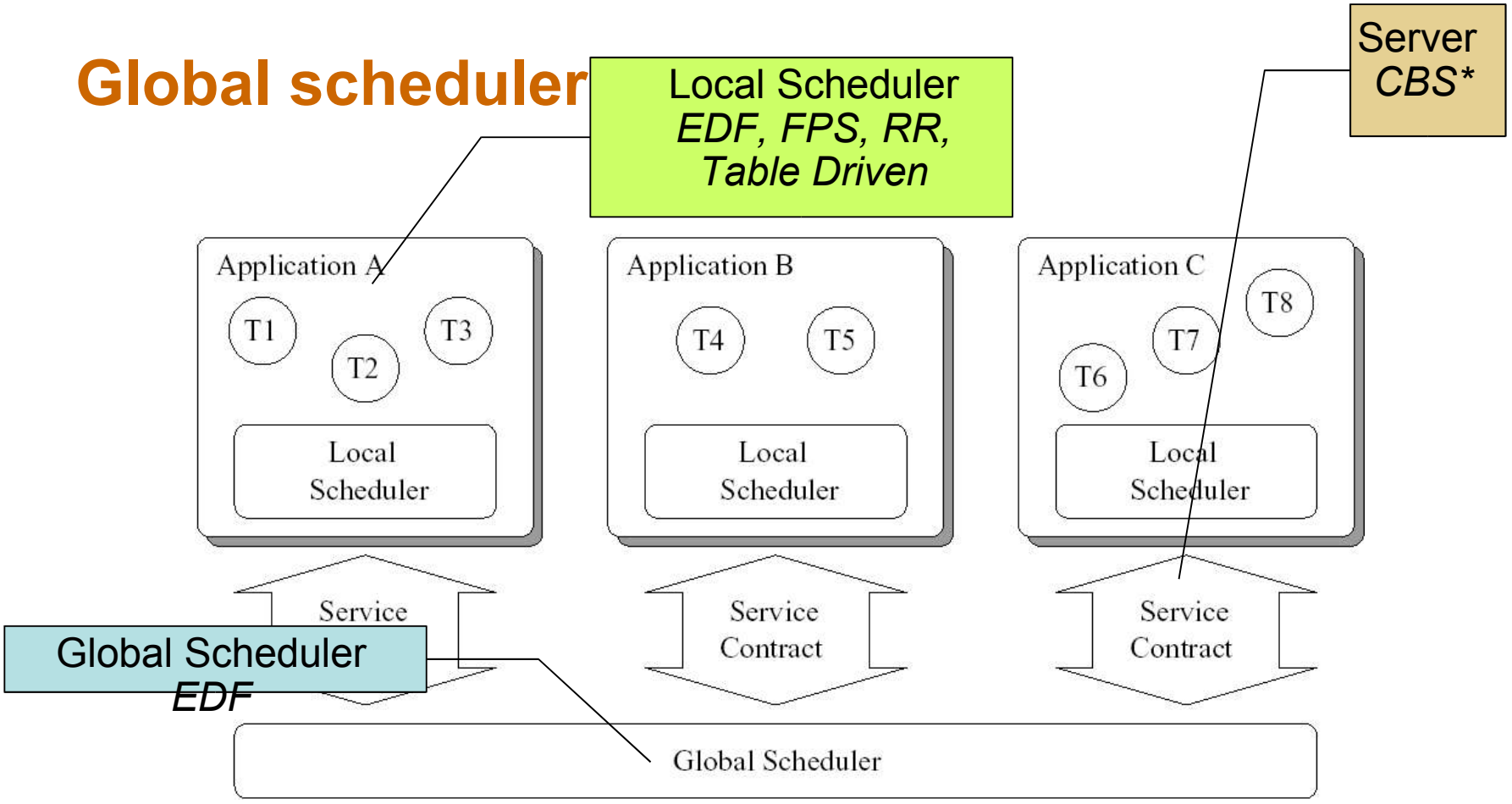


Basic server mechanism: Constant Bandwidth Server (CBS)

Basic scheduling mechanism: EDF

Negotiation mechanism: polynomial schedulability test

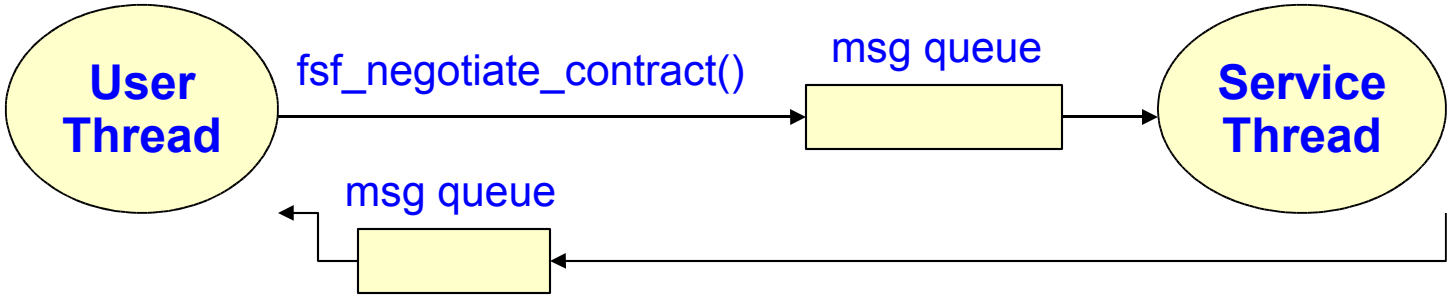
Global scheduler



No implementation-specific data is needed

- no preemption level for servers (automatically assigned by EDF)
- preemption level for resources is not needed either

Contract Negotiation

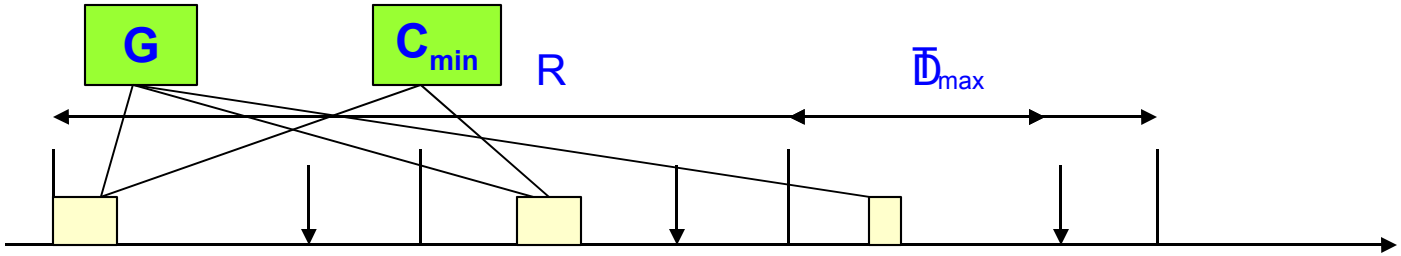


Client/server structure:

the service thread is assigned a contract response time:

- Service thread contract params: (C_{min}, T_{max}, D)
- Computation time for negotiation: G

$$R = \left\lceil \frac{G}{C_{\min}} \right\rceil T_{\max} + D$$



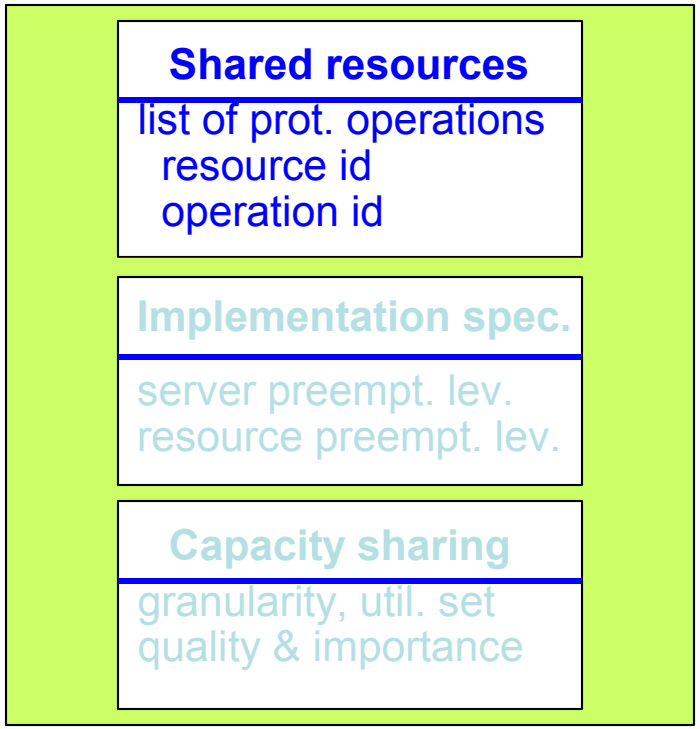
Synchronization

Synchronization between servers

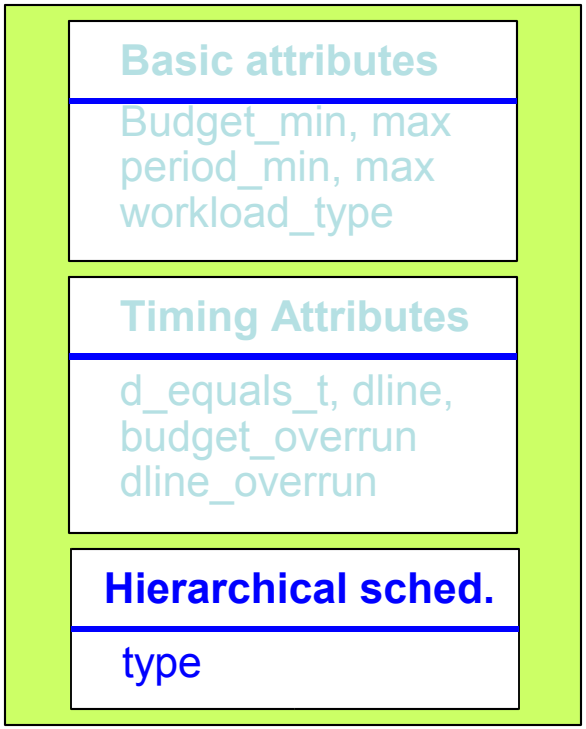
- When two threads in two different servers use the same mutex

Synchronization mechanism

- we are using a mechanism called BWI (Bandwidth inheritance)
- Similar to Priority Inheritance



Hierarchical scheduling



Current support for

- **Fixed Priority (POSIX std)**
- **Round Robin (POSIX std)**
- **EDF**
- **Table Driven (with deadline transformation)**

Easy to introduce new schedulers

- **thanks to Shark modularity**
- **not part of the API**

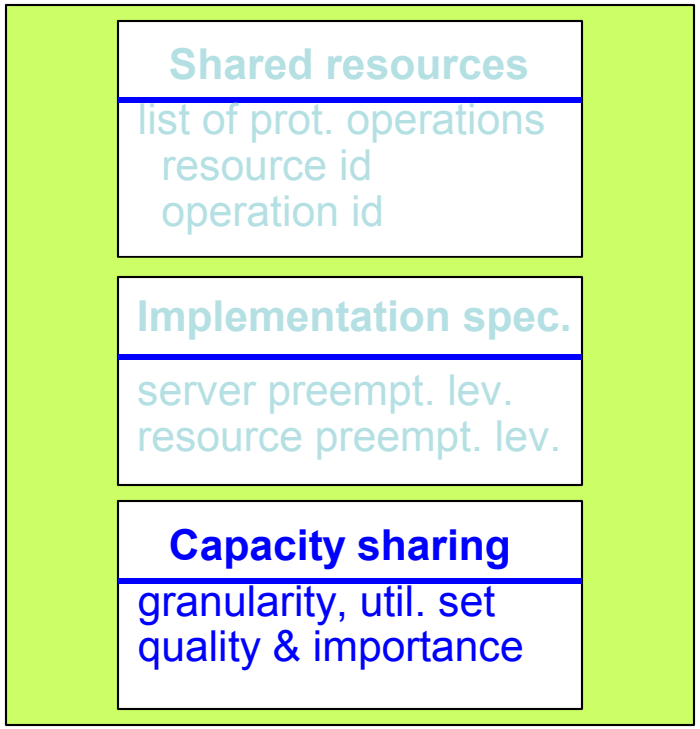
Capacity sharing

The contract is flexible

- **possibility of re-negotiation**
- **possibility of obtaining more than the minimum**

In Shark

- **Use elastic task (Buttazzo et al.) to assign spare capacity**
- **Among those with equal importance, the quality parameter is used as the elastic constant**



Not completed:

- this feature is to be used only when D=T
- planning extension to general model in next phase

Dynamic reclaimation

If some thread execute less than expected, the spare capacity is dynamically reassigned

Current implementation

- the GRUB (greedy reclaimation of unused bandwidth) has been implemented in Shark
- it is not possible to specify which thread gets the extra capacity
- no parameter in the interface

To be done

- the algorithm is valid if D=T
- to be extended to the general model

Examples of usage of the API

Example 1: Initialize a contract for single thread

```
fsf_contract_parameters_t contract;
fsf_server_id_t server;
pthread_t j;

fsf_initialize_contract(&contract);
fsf_set_contract_basic_parameters(&contract,&cmin,&tmax,
                                  &cmax,&tmin,workload);
fsf_set_contract_timing_requirements(&contract,FALSE,&deadline,0,
                                     no_sigval,0,no_sigval);
if (!fsf_negotiate_contract(&contract,&server)) {
        // ERROR
}
else fsf_create_thread(server,&j,NULL,task,NULL,NULL);
```

Define a contract

Initizalize the contract

Create the thread and bind it to the server

Negotiate

Example: typical thread structure

Periodic thread

```
void task_body(void *arg) {
  struct timespec      acttime;
  struct timespec      budget;
  struct timespec      period;
  bool                 deadline_missed;
  bool                 budget_overrun;
  int                  uperiod;

  [...]
  sys_gettime(&acttime);

  while(1) {
        ADDUSEC2TIMESPEC(uperiod,&acttime);
        fsf_schedule_next_timed_job(&acttime,&budget,&period,
                &budget_overrun,&deadline_missed);
        /* Body */
  }
}
```

Example: typical thread structure (2)

Aperiodic thread

```
void task_body(void *arg) {

  fsf_synch_object_handle_t synch_handle;

  struct timespec       budget;
  struct timespec       period;
  bool                  deadline_missed;
  bool                  budget_overrun;

  [...]
  while(1) {

    fsf_schedule_next_event_triggered_job(&synch_handle,&budget,
          &period,&budget_overrun,&deadline_missed);

    /* Body */
  }
}
```

Synchronization object

Wait for next synch. event

Example: hard real-time periodic threads

What is needed

Core service (+ Shared resource synchronization)

Contract Parameters

C_{min}=C_{max}= WCET of the thread

T_{min}=T_{max}= thread's period

D = thread's deadline

workload = bounded

budget overrun exception handling

Advantages

The thread is protected from the other non-RT and soft RT threads in the system (temporal isolation)

if dynamic reclaimation, the spare capacity of this thread can be given to others

Example: soft real-time periodic threads

What is needed

Core + (capacity sharing) + (dynamic recl.) + (shared res. synch.)

Contract Parameters

C_{min} – C_{max} = variation of the execution time

T_{min}=T_{max}= thread's period

D = thread's deadline

workload = indeterminate

Advantages

Does not impact on other threads (temporal isolation)

minimum service is guaranteed

Takes advantage of capacity sharing and dynamic reclamation (to minimize deadline misses)

can re-negotiate if it needs more

Example: imprecise computation

Thread consists of a mandatory part and N optional parts

- WCET of mandatory part = M
- WCET of i-th optional part = O_i

What is needed

Core + (capacity sharing) + (dynamic recl.) + (shared res. synch.)

Contract Parameters

$C_{min} = M$

$C_{max} = M + O_1 + ... + O_N$

$T_{min} = T_{max}$ = thread's period

D = thread's deadline

workload = bounded

Example: thread structure of an imprecise computation thread

```
void task_body(void *arg) {
  ...
  pthread_t my_pid = (pthread_t)(arg);
  int i;
  sys_gettime(&acttime);
  fsf_get_server(&server, my_pid);

  while(1) {
    /* Mandatory Body */
    for (i=0; i<N; i++) {
        fsf_get_available_capacity(server, &capacity);
        if (capacity > O[i]) {
         /* Optional Code */
        }
        else break;
    }
    ADDUSEC2TIMESPEC(uperiod,&acttime);
    fsf_schedule_next_timed_job(&acttime,&budget,&period,
                  &budget_missed,&deadline_missed);
  }
}
```

Get remaining capacity

If enough capacity execute optional part

Example: imprecise computation

Advantage:

- mandatory part is guaranteed
- If capacity sharing and dynamic reclamation services are available, some optional part may be completed as well
- More reclaiming  more optional parts

Example: creating a contract for an application

What is needed

Core + Hierarchical

Contract Parameters

C_{min} – C_{max}

T_{min} – T_{max}

D

use Lipari & Bini method (ECRTS 03)

workload = indeterminate

Scheduler type = EDF or FPS or RR or TD

Advantages

re-using an existing code base without re-designing and re-implementing it

need minimal modifications to the original code

Example: contract for hierarchical

```
fsf_contract_parameters_t contract;
fsf_server_id_t server;
pthread_t j,k;

HARD_TASK_MODEL ht;

fsf_initialize_contract(&contract);
fsf_set_contract_basic_parameters(&contract,&cmin,&tmax,
                          &cmax,&tmin,workload);

fsf_set_contract_timing_requirements(&contract,FALSE,&deadline,0,
                          no_sigval,0,no_sigval);

fsf_set_local_scheduler_parameter(&contract,FSF_SCHEDULER_EDF);

fsf_negotiate_contract(&contract,&server);

/* S.Ha.R.K. hard task parameters */
hard_task_default_model(ht);
hard_task_def_mit(ht,TIMESPEC2USEC(&deadline));
hard_task_def_wcet(ht,TIMESPEC2USEC(&wcet));

/* Create EDF task */
fsf_create_local_thread(server,&j,NULL,task,NULL,&ht);

/* Create EDF task */
fsf_create_local_thread(server,&k,NULL,task,NULL,&ht);
```

Set scheduler

Create a local thread