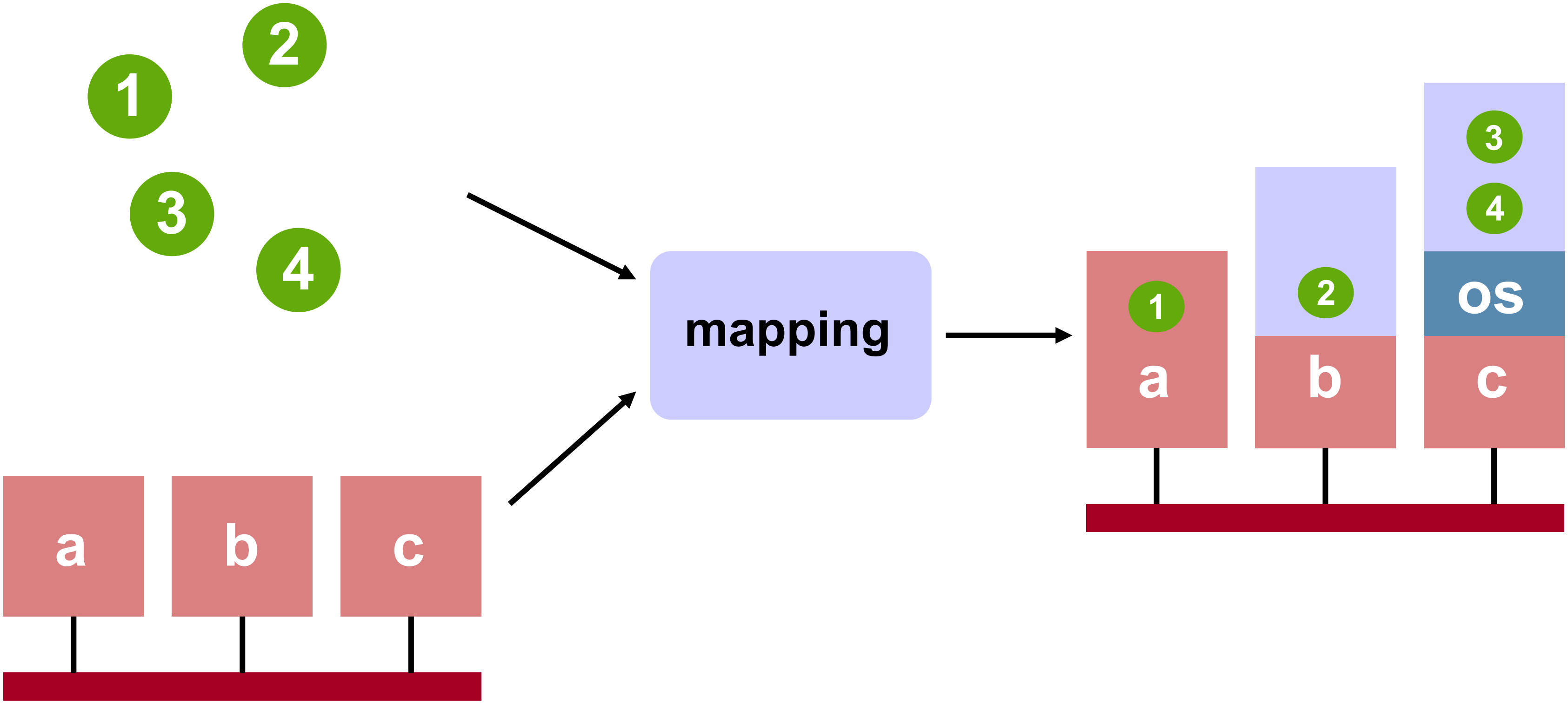
Abstract RTOS Modelling for Multi-Processor SoC using SystemC

Prof. Jan Madsen

Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, Building 321
DK2800 Lyngby, Denmark
jan@imm.dtu.dk

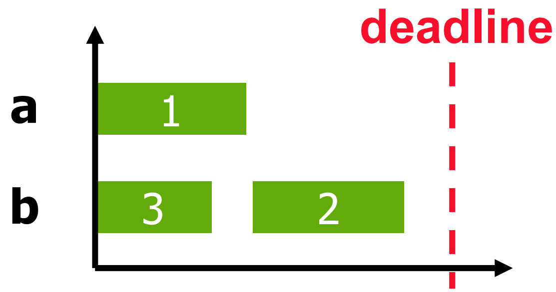
Funded by SoC-Mobinet (IST 2000-30094)

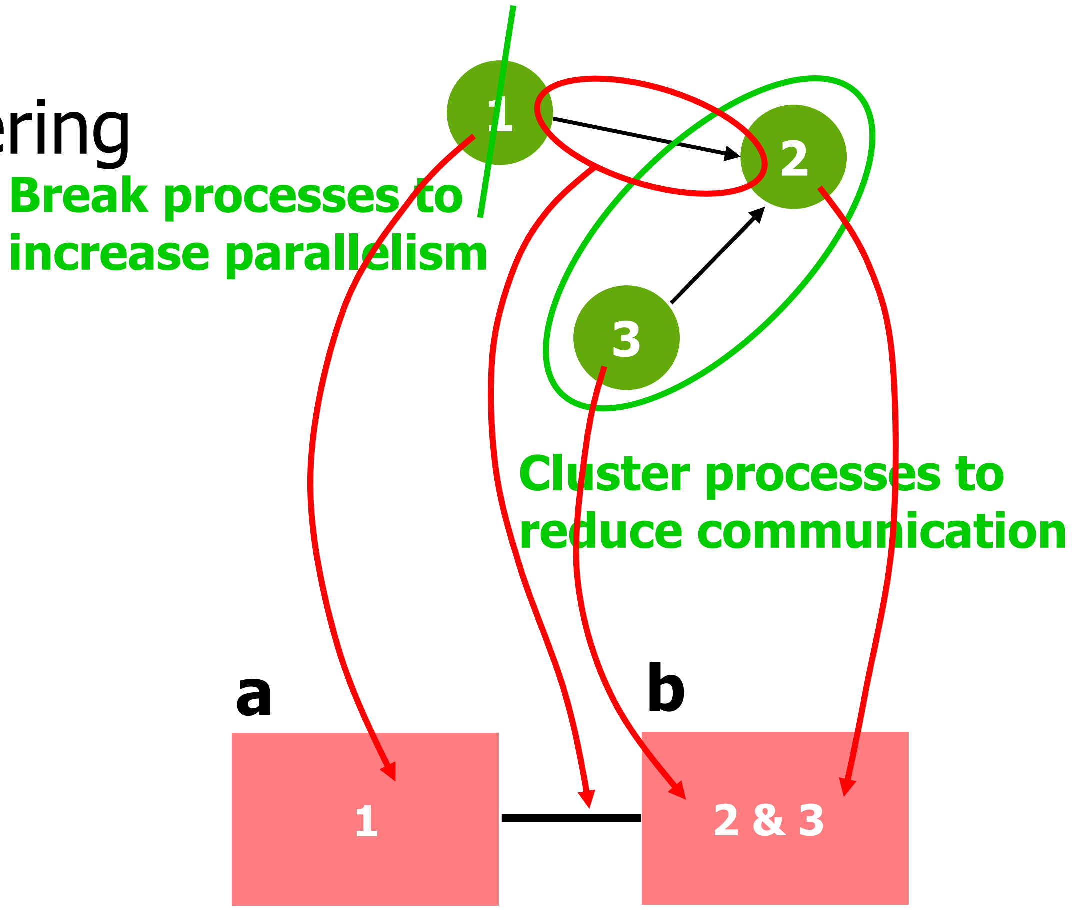
Motivation



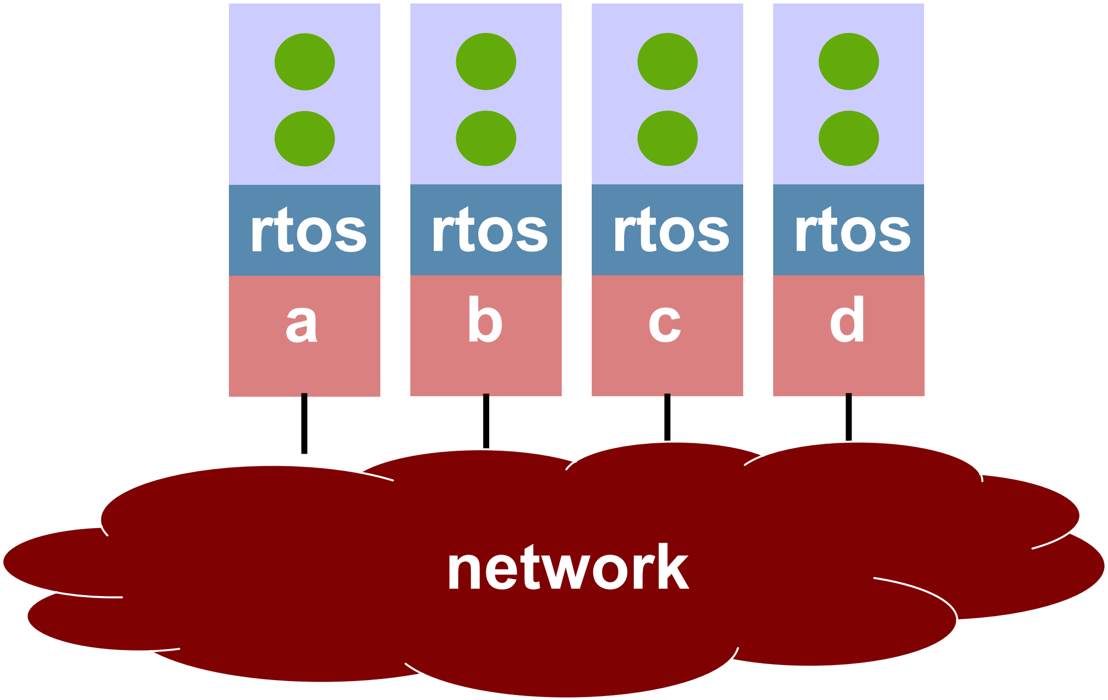
Principles of mapping

- Partitioning/clustering
- Allocation
- Mapping
- Scheduling
- Communication

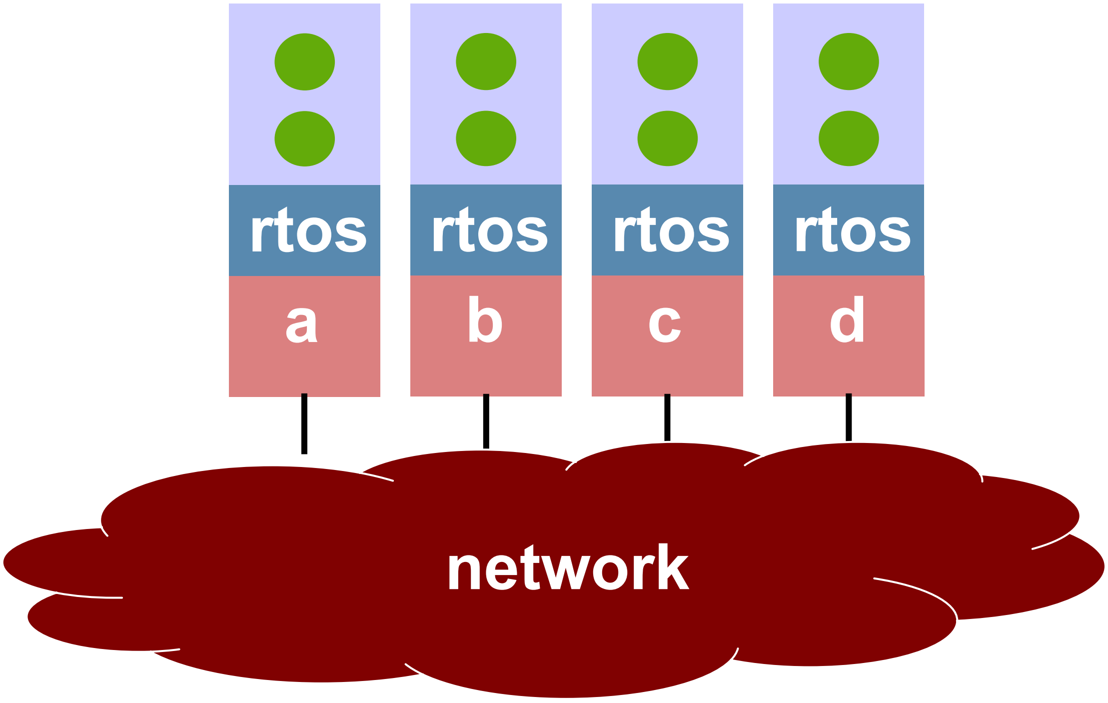




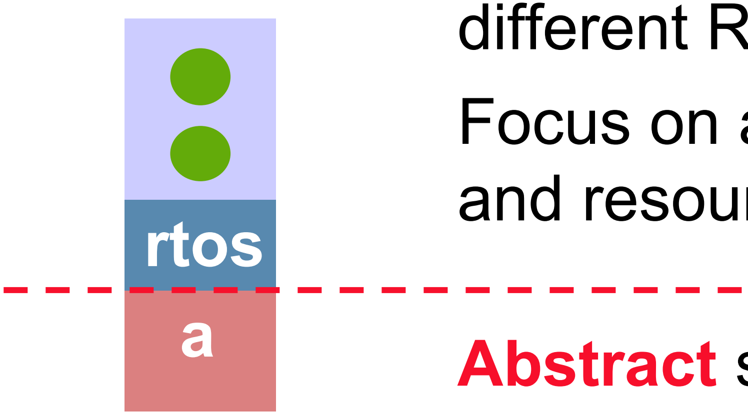
Motivation



Uni-processor ...



Uni-processor ...



Framework to experiment with different RTOS strategies

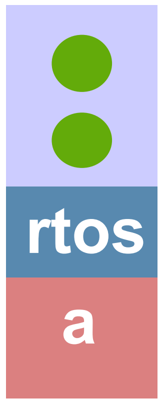
Focus on analysis of **timing** and resource sharing

Abstract software model, i.e. no behavior/functionality

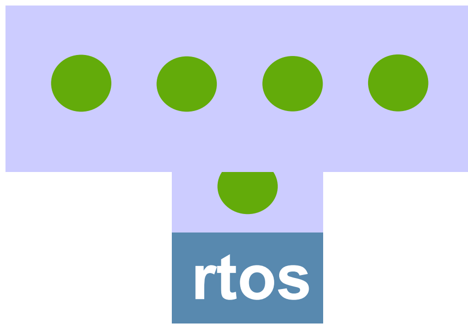
Easy to create tasks and implement RTOS models

Based on **SystemC**

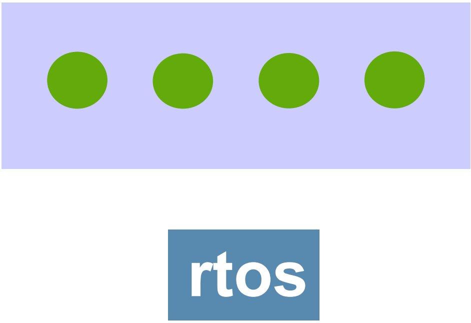
System model



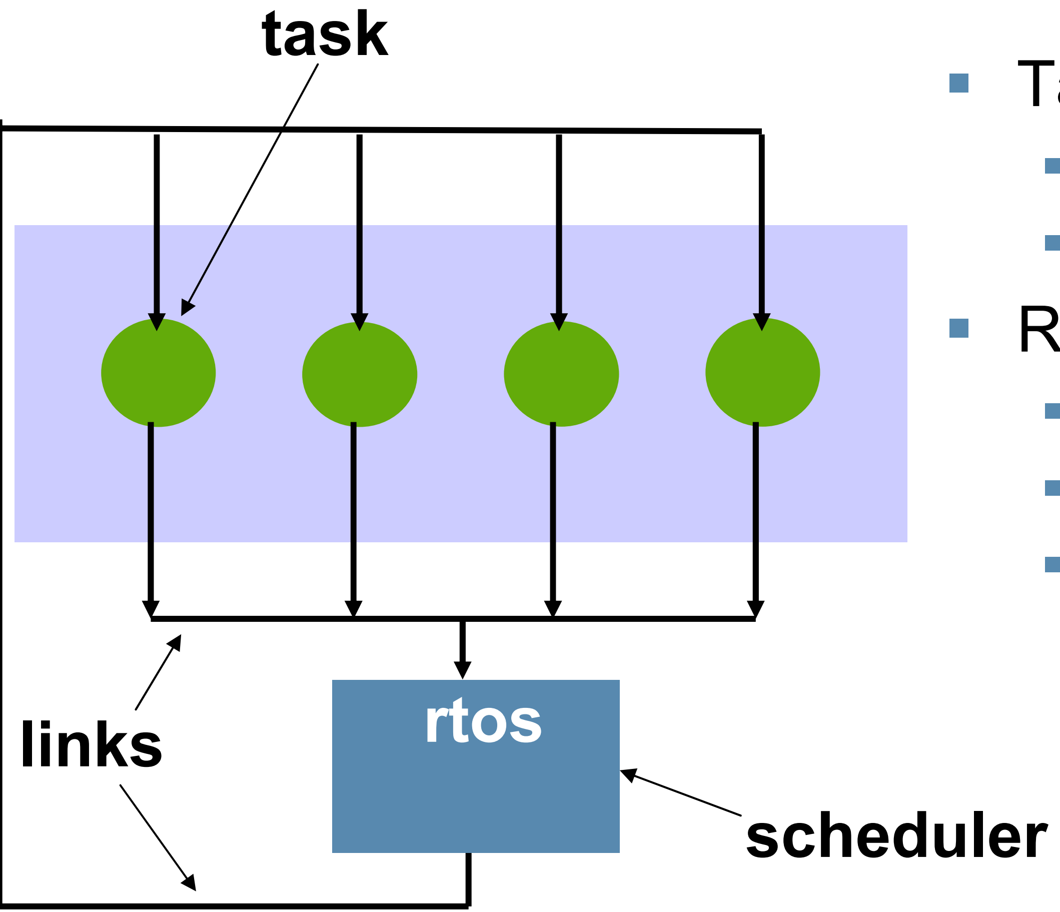
System model



System model

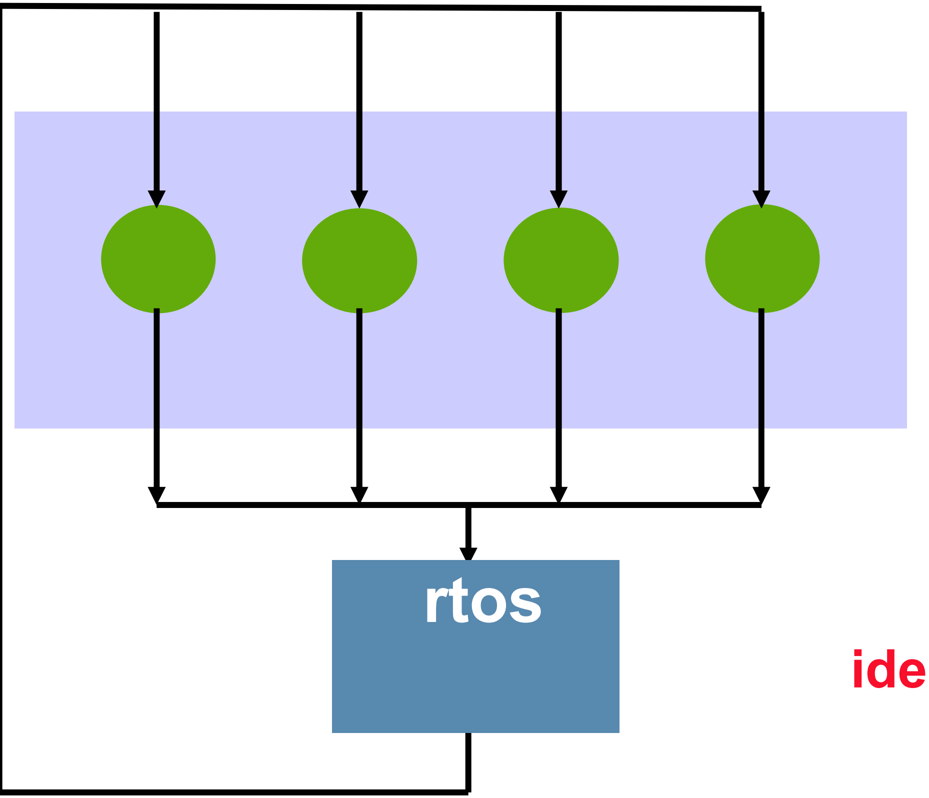


System model



- Task messages:
  - **`ready`**
  - **`finished`**
- RTOS commands:
  - **`run`**
  - **`preempt`**
  - **`Resume`**

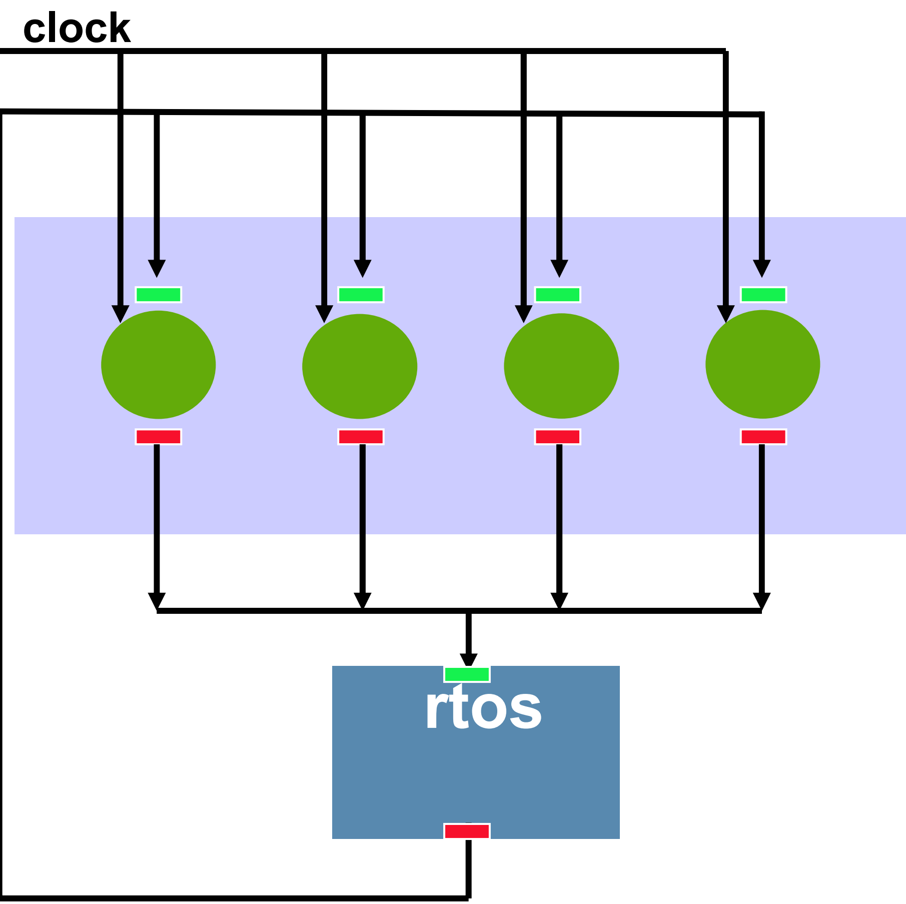
System model - SystemC




```
pa = new
   task("task_a",1,50,3,12,0,ready);
registerTask(pa);

pb = new
   task("task_b",2,40,2,10,0,ready);
registerTask(pb);

pc = new
   task("task_c",3,30,1,10,0,ready);
registerTask(pc);
```

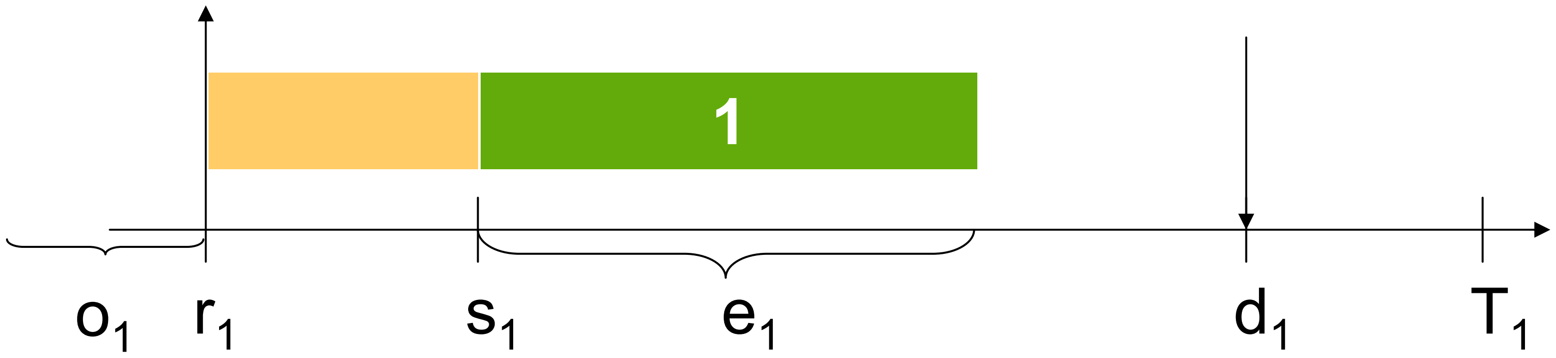
identifier period priority WCET offset

Link model



- Aim: Adding tasks without having to create seperate communication links
- Uses the SystemC master-slave library
- If two tasks send a message at the same time – they are executed in sequence, but in undefined order
- Global ”clock” is used to keep track of time

Task model



r_1 = time at which task becomes ***released (or active)***

s_1 = time at which task ***starts*** its execution

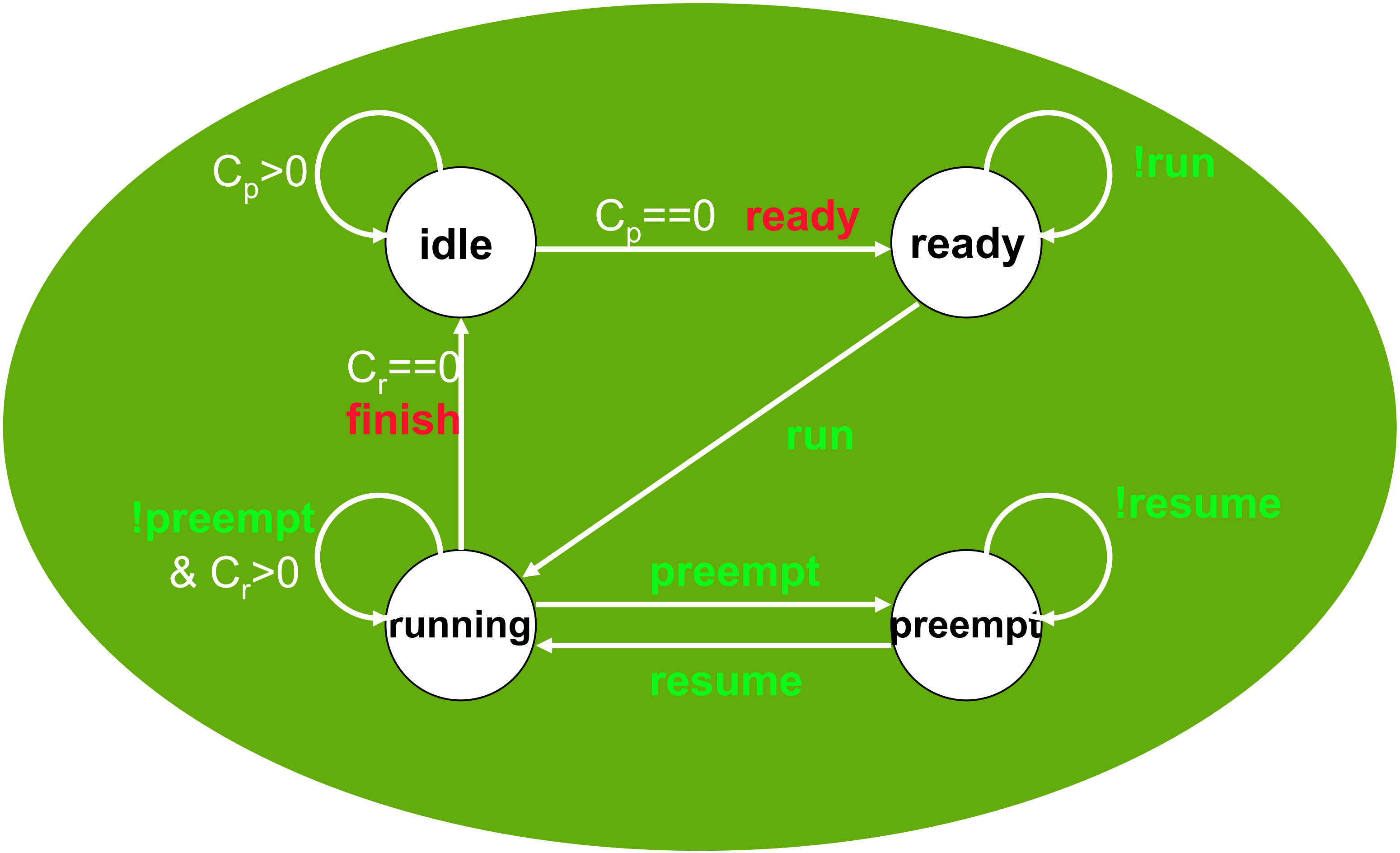
e_1 = worst case ***execution*** time (WCET)

d_1 = ***deadline***, task should complete before this!

T_1 = ***period***, minimum time between task releases

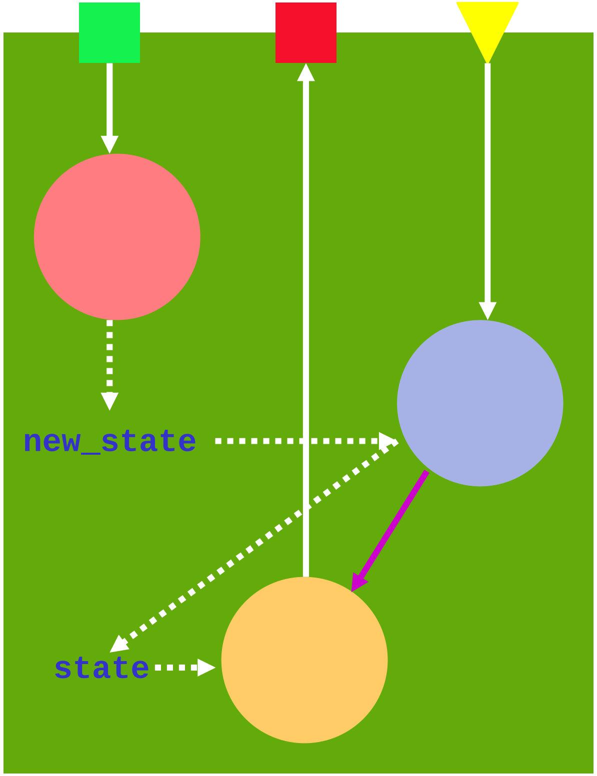
o_1 = ***offset (or phase)*** for first released

Task model

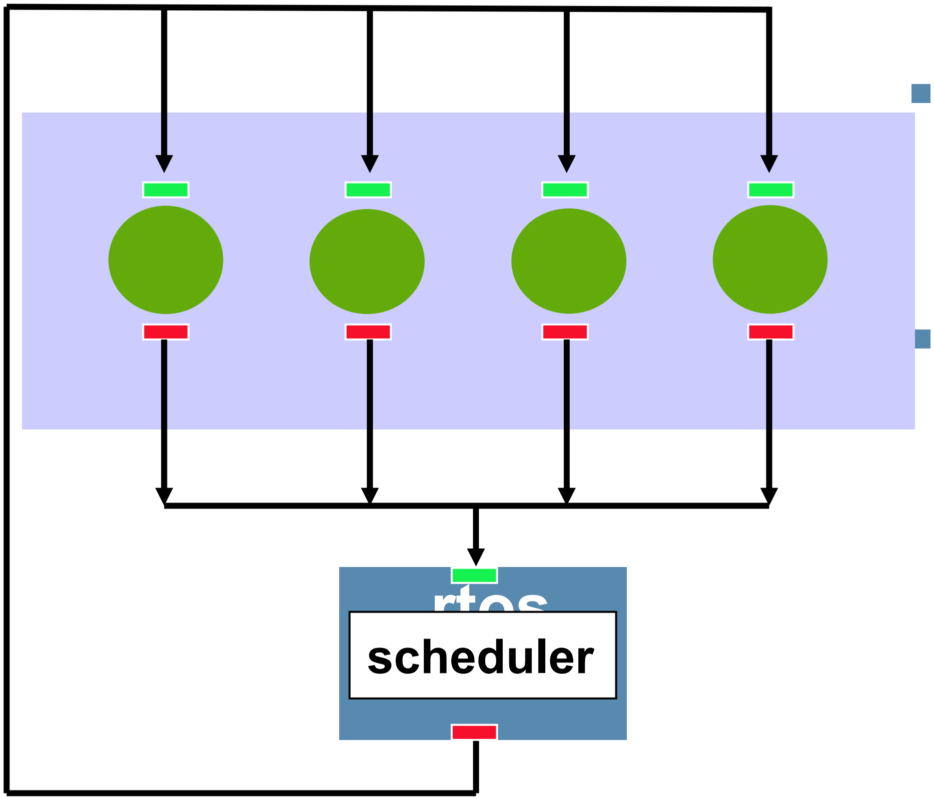

$C_p>0$
idle
$C_p==0$ ready
ready
!run
$C_r==0$
finish
run
!preempt
& $C_r>0$
!resume
preempt
running
preempt
resume

Task model - SystemC

```
SC_MODULE(perTask){

   sc_outmaster<message_type>  out_message;
   sc_inslave<message_type>    in_command;
   sc_in_clk                   clock;

   sc_event                    newStateEvent;

   void next_state();
   void update_state();
   void get_comm_from_scheduler();

   SC_HAS_PROCESS(perTask);

   perTask(sc_module_name name_, int id_, ...)
          : sc_module(name_),id(id_), ...)
   {
       SC_METHOD(next_state);
       sensitive << newStateEvent;

       SC_METHOD(update_state);
       sensitive << clock;

       SC_SLAVE(get_comm_from_scheduler,in_command);
   }
   private:
       t_state  state, new_state;
```



Scheduling model



- Aim: Simple way to describe the scheduling algorithm
- Scheduler should only handle one message at a time
- Rate-monotonic scheduling
  - preemptive
  - WCET
  - d=T
  - Fixed priority

Scheduling model - SystemC

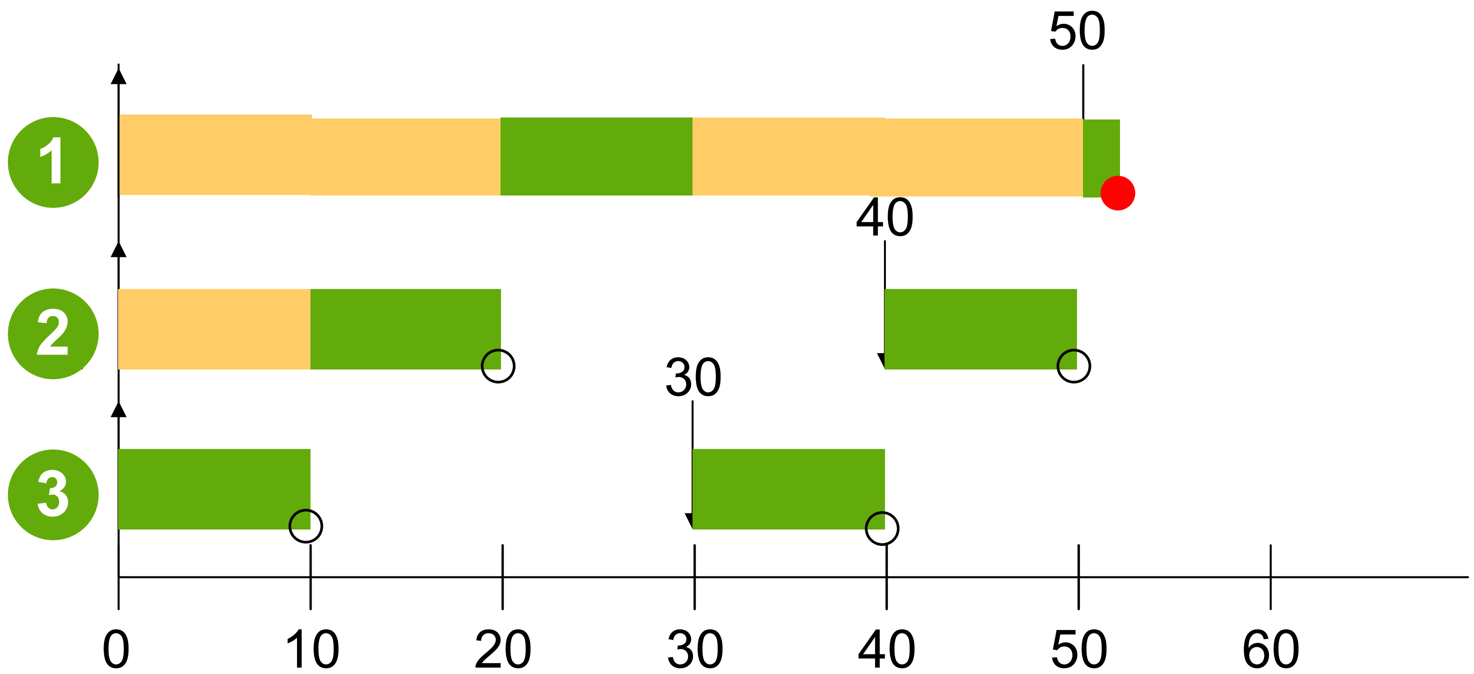
Rate monotonic scheduling

```
void RM_scheduler::doSchedule() {
  ti = in_message;
  if (ti.comm==ready) {
    Q.push(ti);
  } else {
    tk.id = 0;
  }
  tj = Q.top();
  if (tj.id != 0) {
    if (tk.id != 0) {
      if (tk < tj) {
        out_command = *(preemptTask(&tk, &Q));
        tk = tj;
        out_command = *(runTask(&tj, &Q));
      } else {
      }
    } else {
      tk = tj;
      out_command = *(runTask(&tj, &Q));
    }
  } else {
  }
}
```

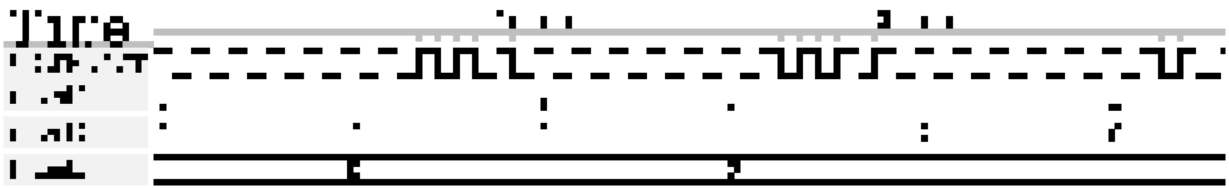
- `ti`: message received from from task `i`
- `tj`: message from task `j` with highest priority from ready list
- `tk`: message of the currently running task

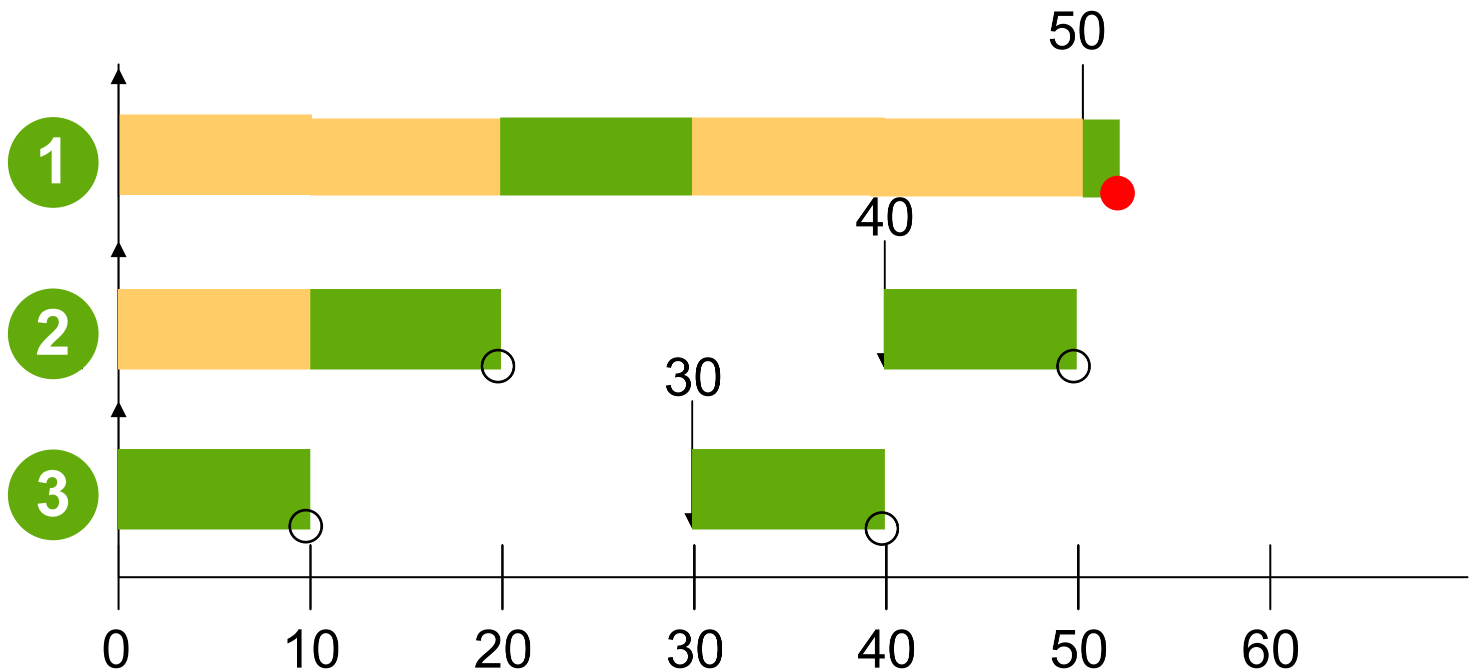
An example

	T_i	e_i	priority
1	50	12	3
2	40	10	2
3	30	10	1



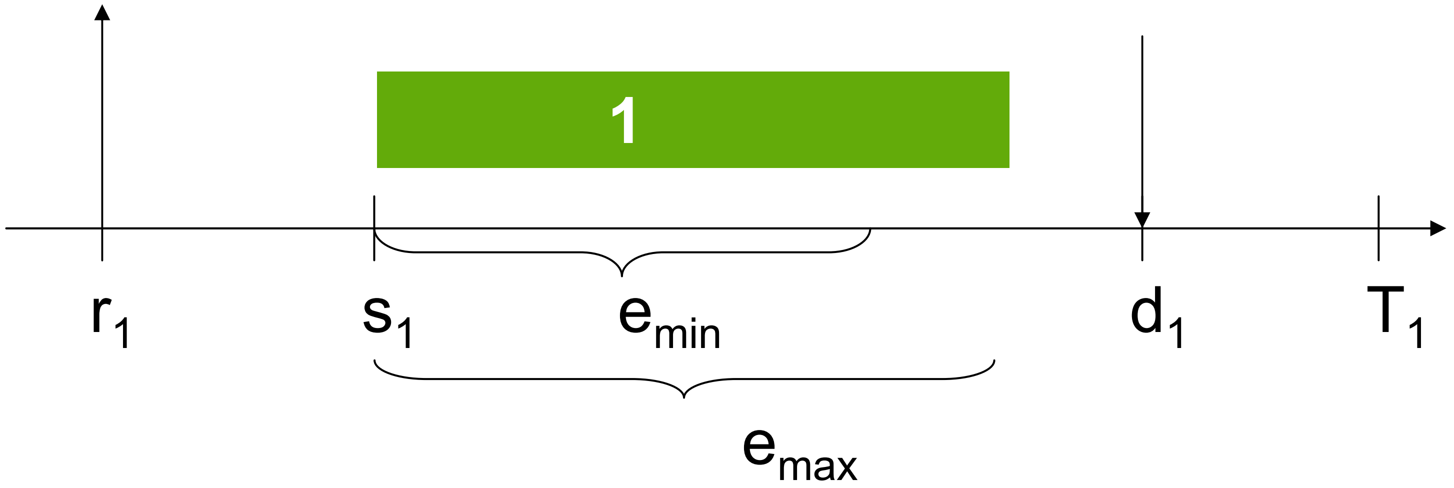
An example





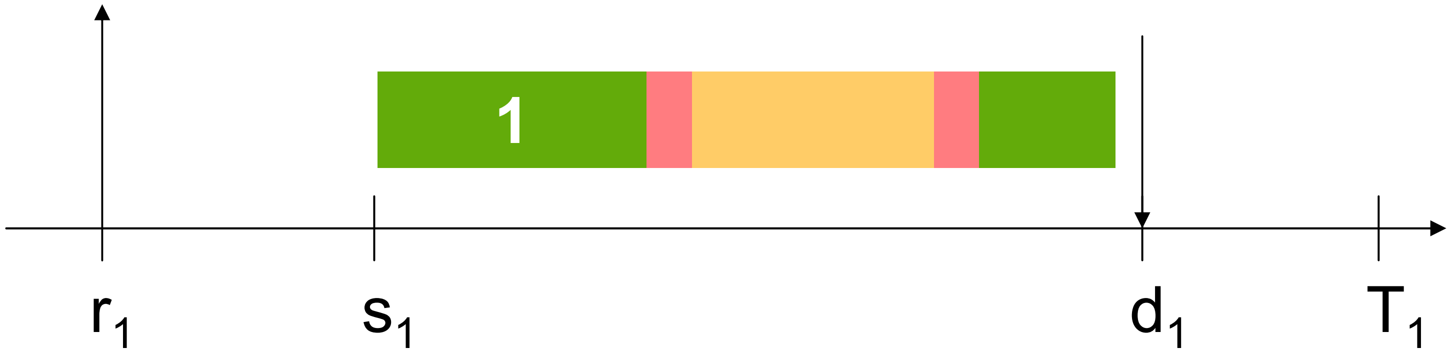
Extending the task model

- Varying execution times $[e_{min}:e_{max}]$

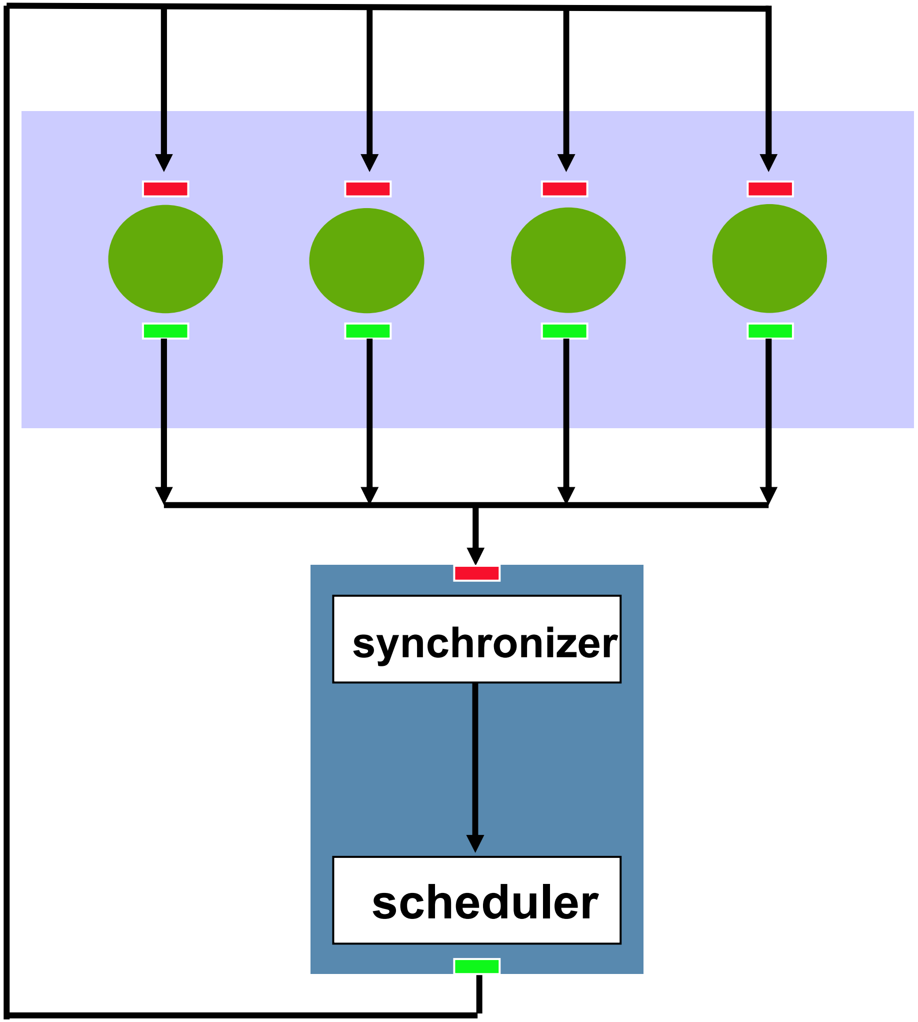


Extending the task model

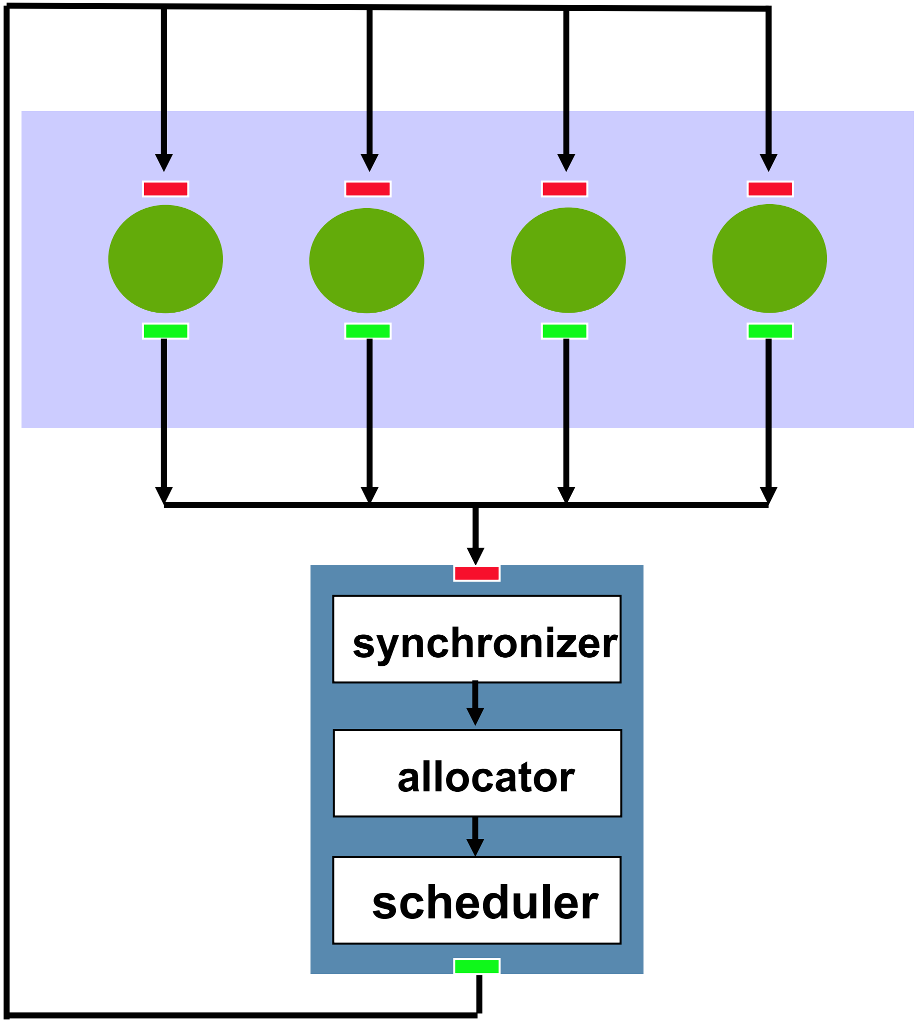
- Varying execution times [e_{min}:e_{max}]
- Context switching
- Data dependencies



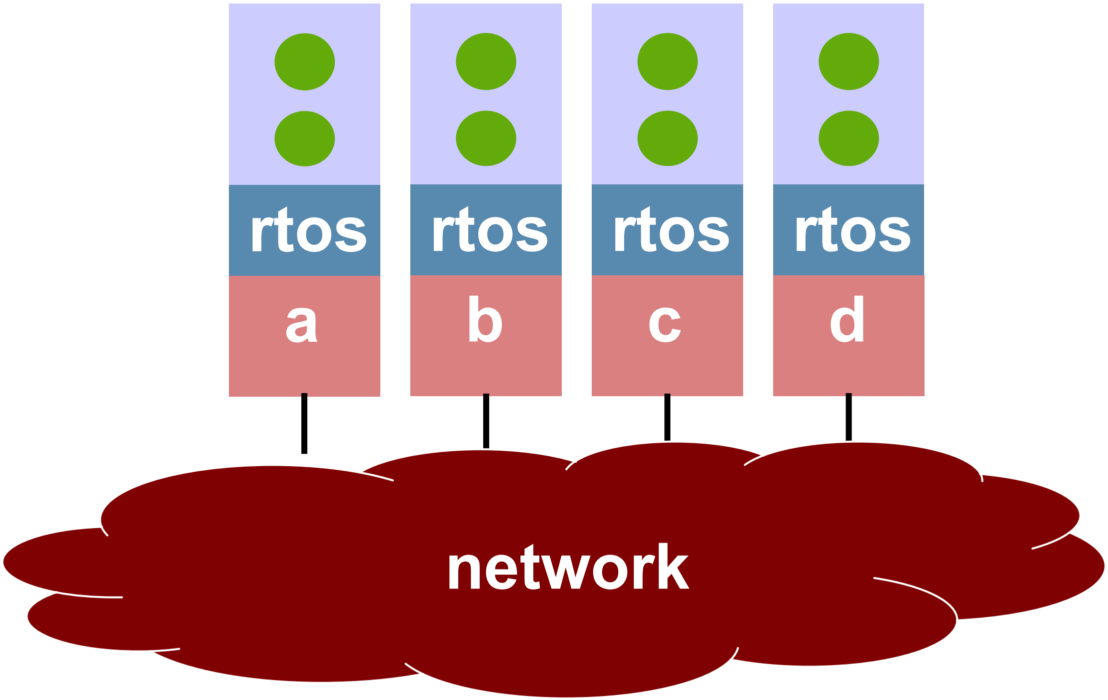
Data dependencies



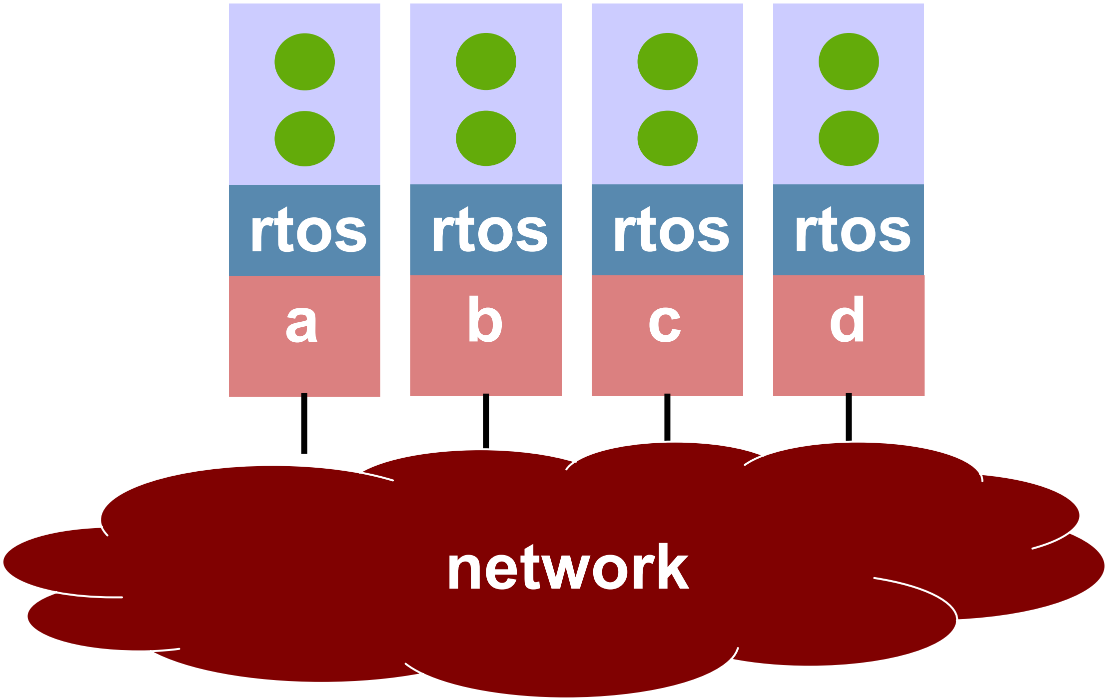
Resource sharing



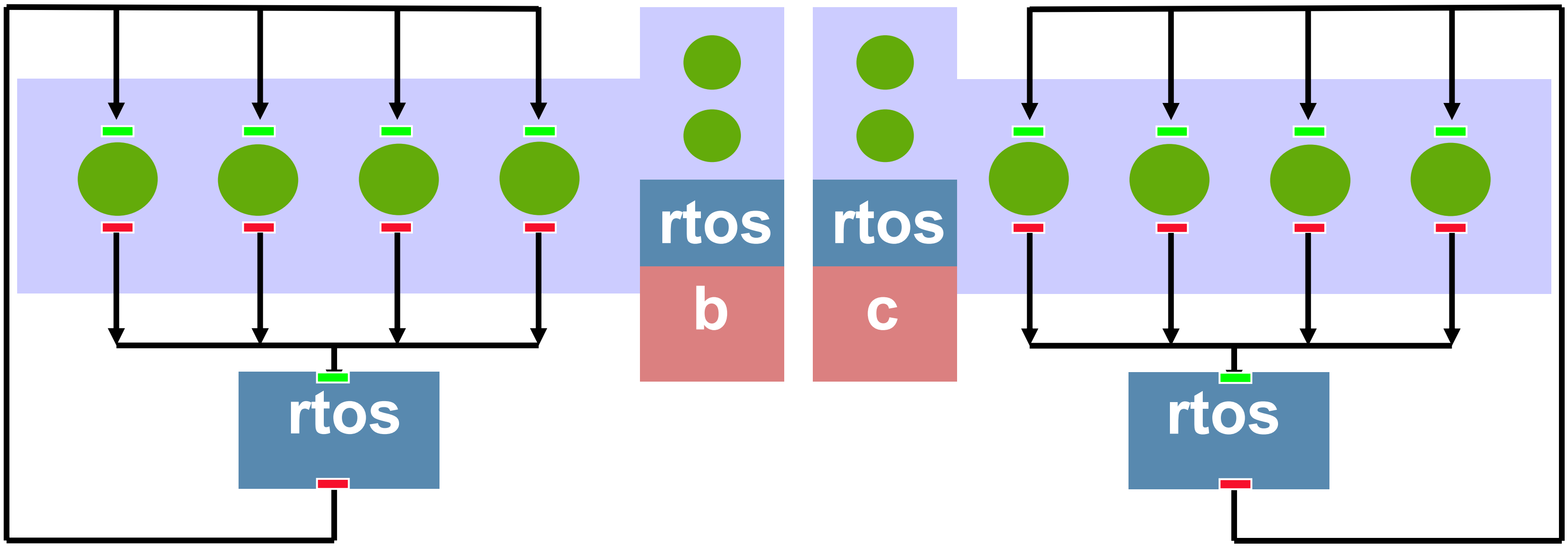
Multi-processors



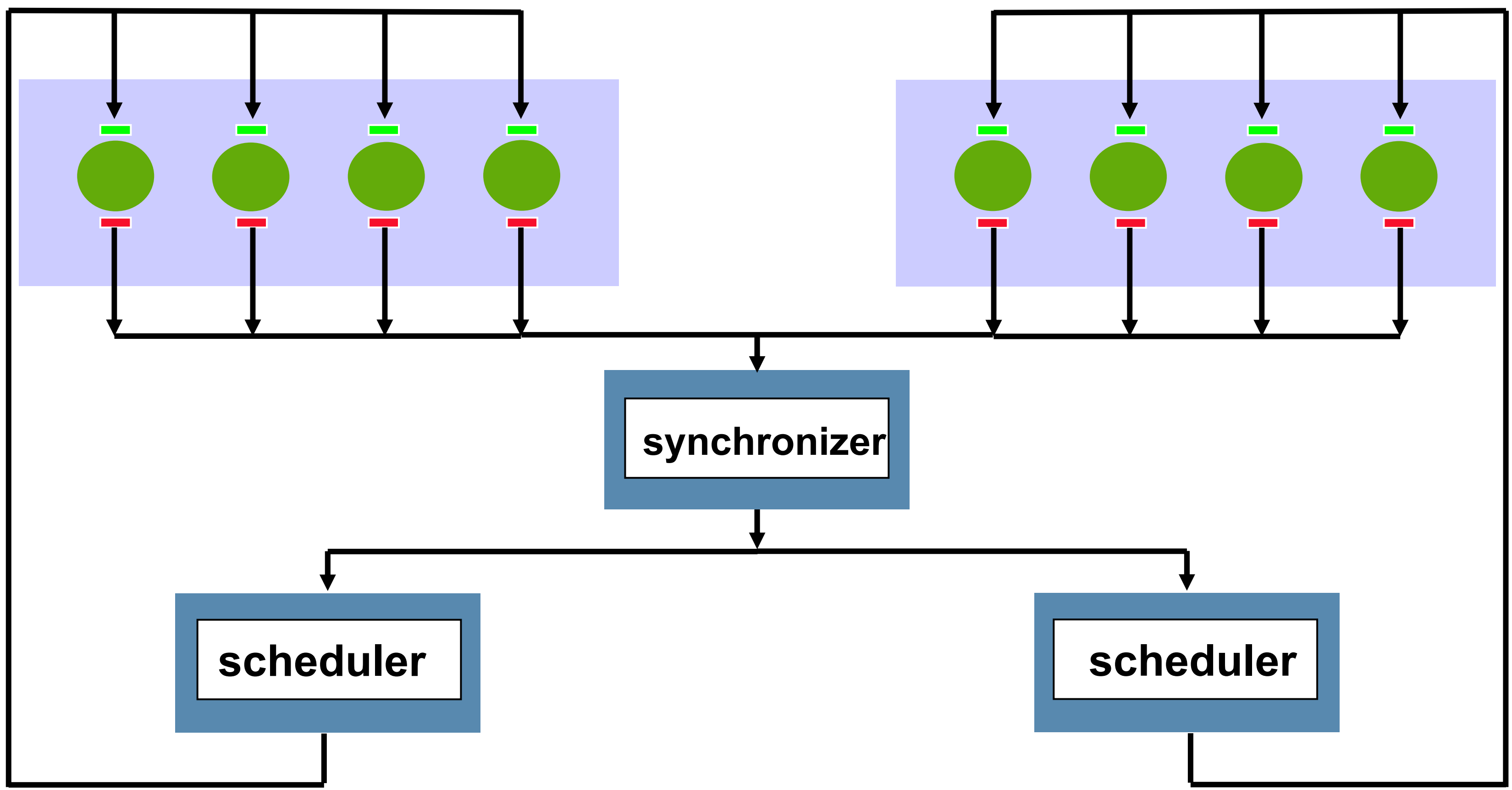
Multi-processors



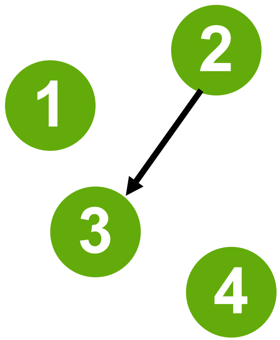
Multi-processors

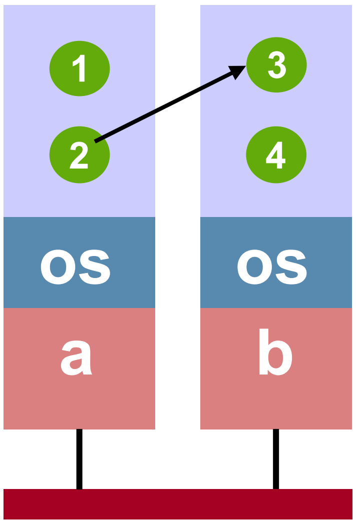


Multi-processors



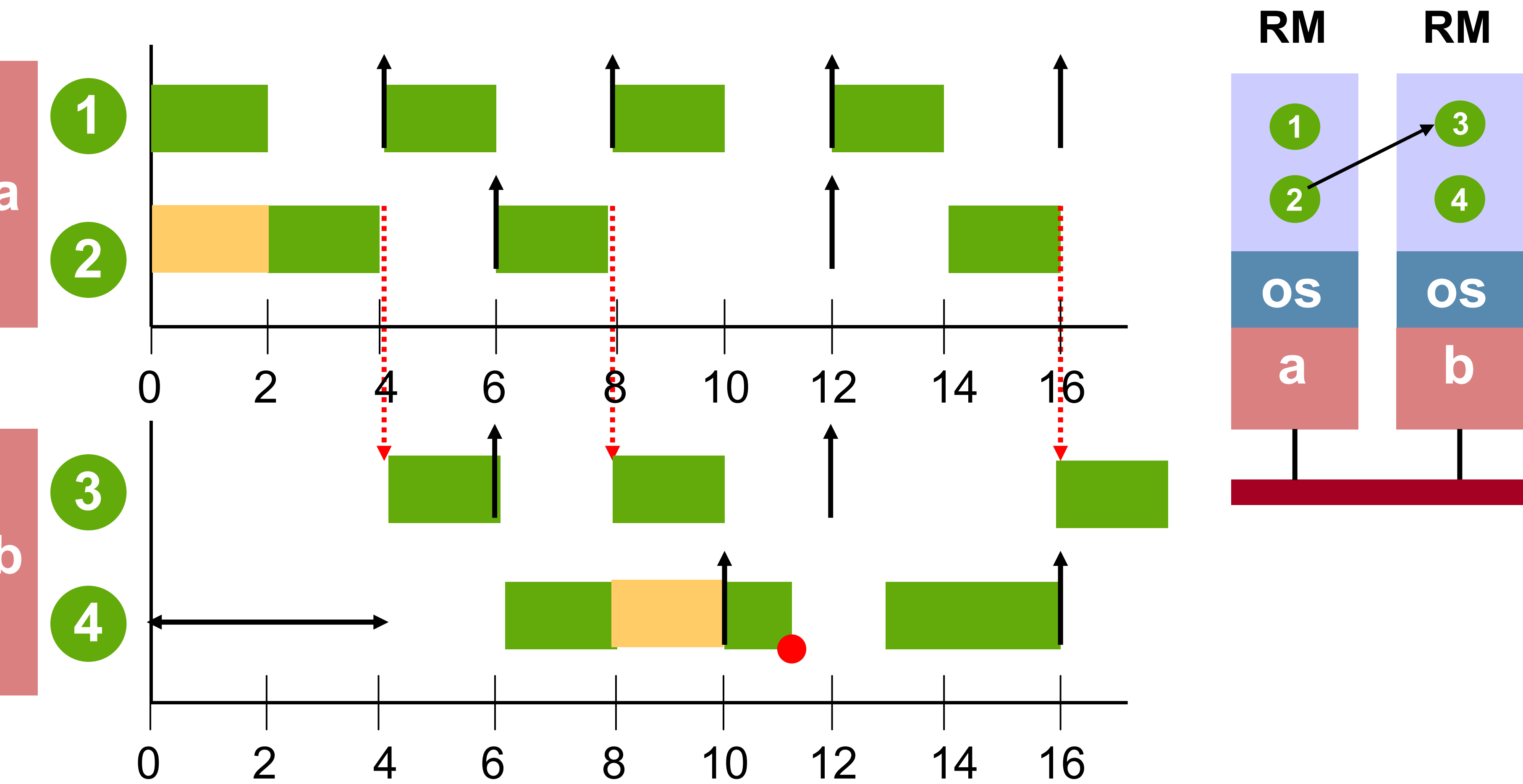
Example



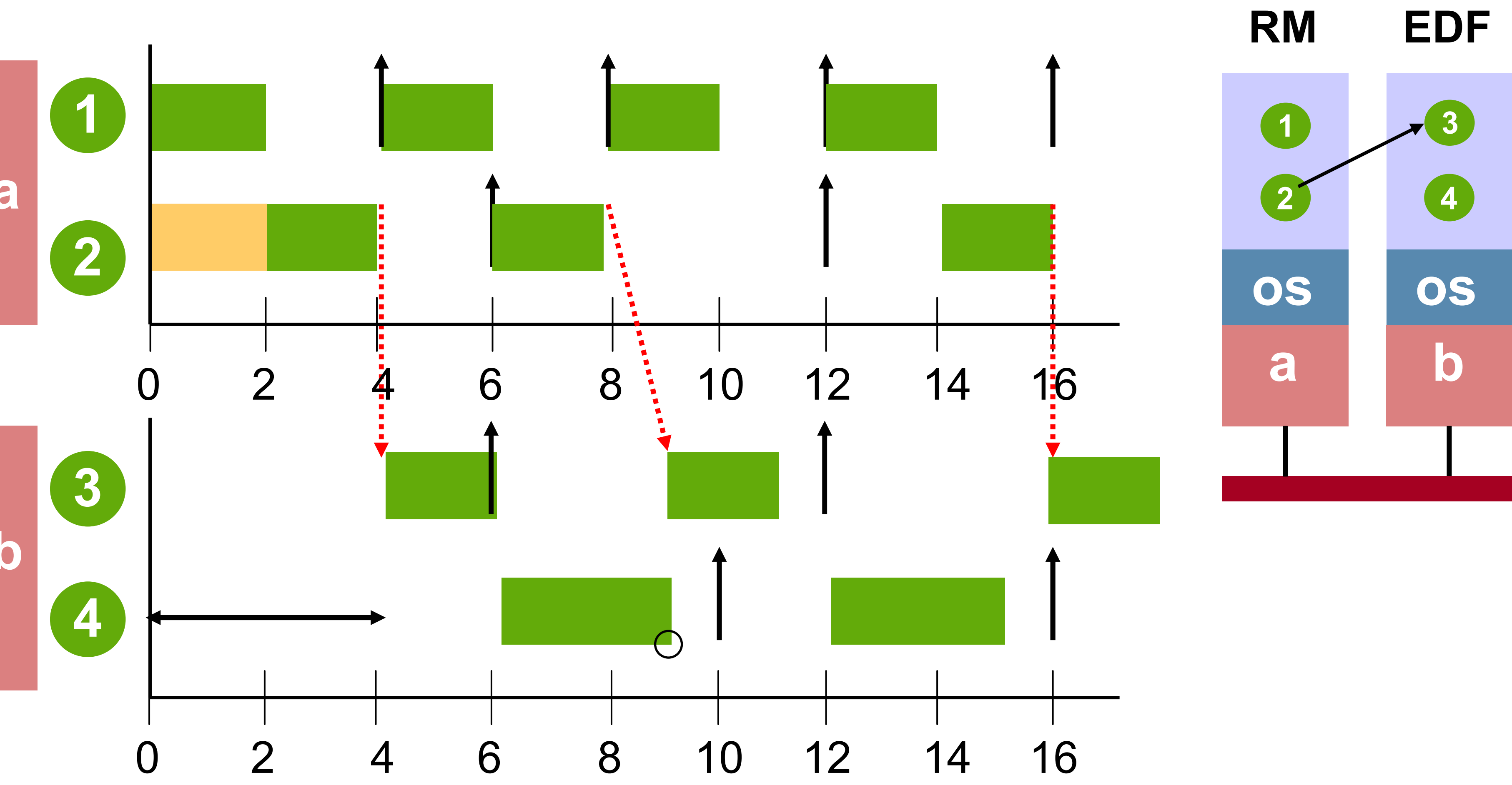


	o_i	T_i	e_i
1	0	4	2
2 → 3	0	6	2+2
4	4	6	3

Dynamic scheduling



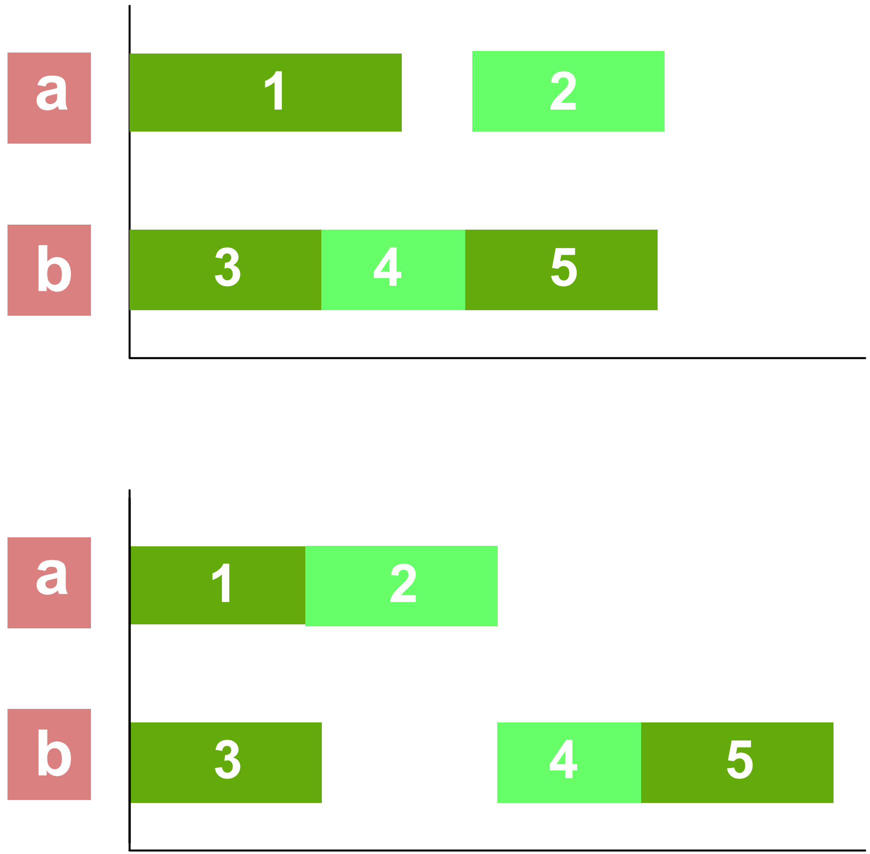
Changing synchronization protocol

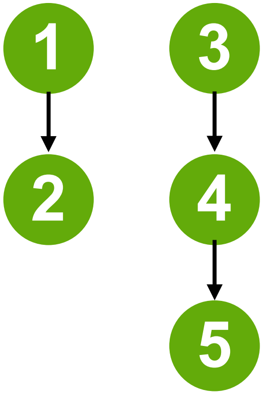


Multi-processing anomalies

- Assume a set of tasks optimally scheduled on a multiprocessor system with:
  - fixed number of processors
  - fixed execution times (e_i)
  - precedence constraints
- Then
  - ***changing*** the priority list
  - ***increasing*** the number of processor
  - ***reducing*** execution times
  - ***weakening*** the precedence constraints
- May ***increase*** the scheduling length!

Example of anomalies





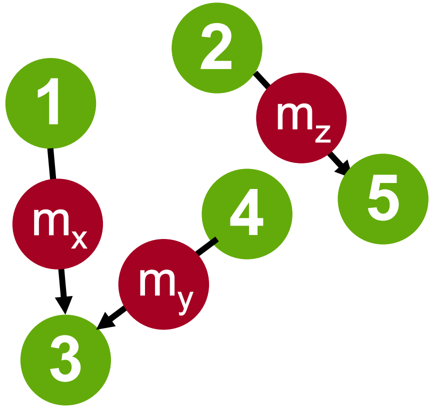
Task 2 and 4 are sharing a resource, i.e. mutually exclusion

Reduce e_1 of task 1

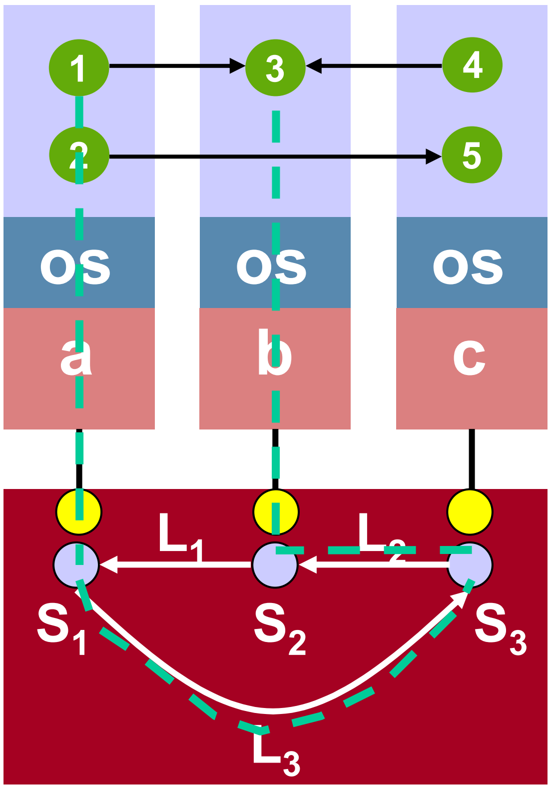
Consequences of anomalies

- Tasks may complete before their WCETs
- So most on-line scheduling algorithms are subject to experience anomalies
- Simple but inefficient solution:
  - Have tasks completing early idle

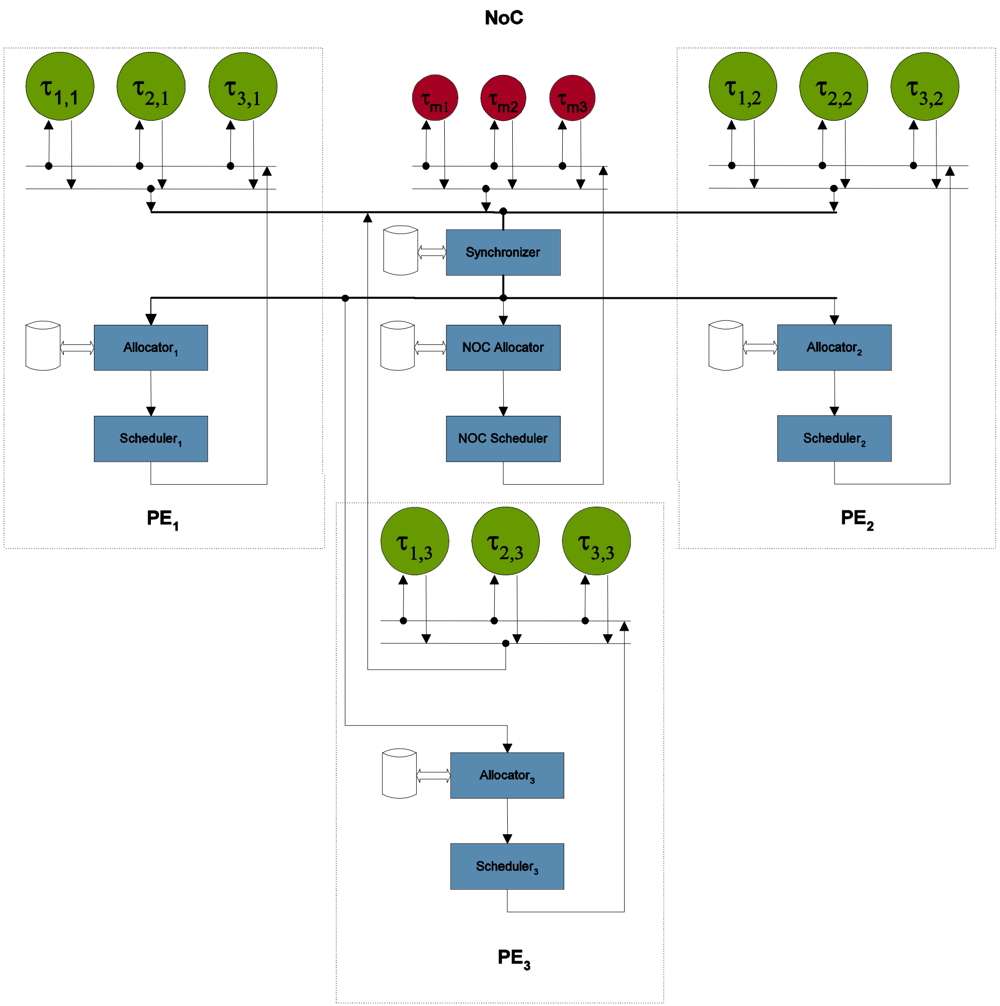
Network-on-Chip extension



m_x: S_1, L_3, S_3, L_2, S_2



Network-on-Chip model

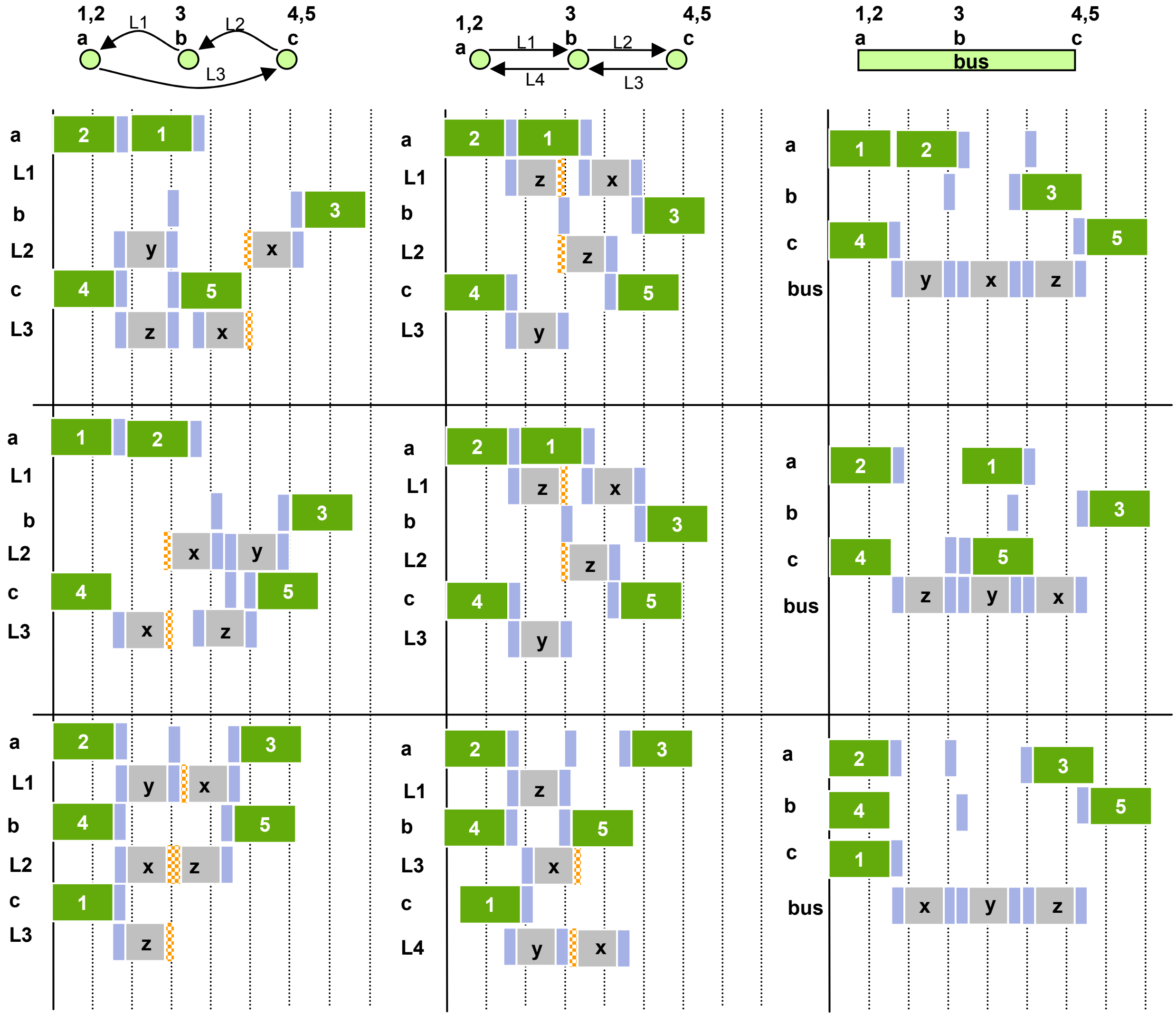


Design space exploration

Timing Aware

QoS Aware
Any traffic from a
has higher priority

Allocation Aware
Swap task on PEs
to: 2,3 | 4,5 | 1



Summary

- Simple SystemC based framework to study the dynamic behavior of a task set running under the supervision of an abstract RTOS
- Synchronizer to handle data dependencies
- Extension to multi-processor/RTOS systems
- Network-on-Chip extension (*new*)
- Not covered,
  - Allocator to handle resource sharing
  - Power estimation/profile
  - Multi-processing anomalies (*in your slides*)

References

- **Scheduling in Real-Time Systems**
  F. Cottet, J. Delacriox, C. Kaiser, Z. Mammeri
  John Wiley & Sons, 2002
- **Real-Time Systems**
  J. W. S. Liu
  Printice Hall, 2000
- **Real-Time Systems and Programming Languages**
  A. Burns, A. Wellings
  Addison-Wesley, 2001 (third edition)
- **System Design with SystemC**
  T. Grotker, S. Liao, G. Martin, S. Swan
  Kluwer, 2002