Lab of EMIoT
Welcome!

Objective and organization

- Logistics: In-class lab
  - 1 laptop per person
  - May be useful to bring portable multiple sockets

- Necessary software:
  - C
  - Any scripting language to do plots, data analytics, … E.g.,
    - MATLAB (You can get a free student licence https://www.areait.polito.it/supporto/risultato_serv.asp?serv=matlab&dettaglio=S&id_progetto_servizio=331)
    - Python
    - Whatever you like…

LAB schedules

- 20% of the final score
  - 9 points maximum

- Assignments will be evaluated
  - Groups of **2 people**
  - 1 report per lab, per group
  - Any extension to the minimum assignment **may** lead to an increase in the evaluation
    - Make sure you meet all requirements
    - Do not go out of topic

LAB delivery

- Lab deadline is 23:59 of the day before the 2nd exam
  - **No exception**
  - Late delivery implies <u>zero score</u> for labs

- Format: **one archive**
  - File name = report.zip
  - One subfolder per lab (Lab1/ Lab2/ Lab3/)

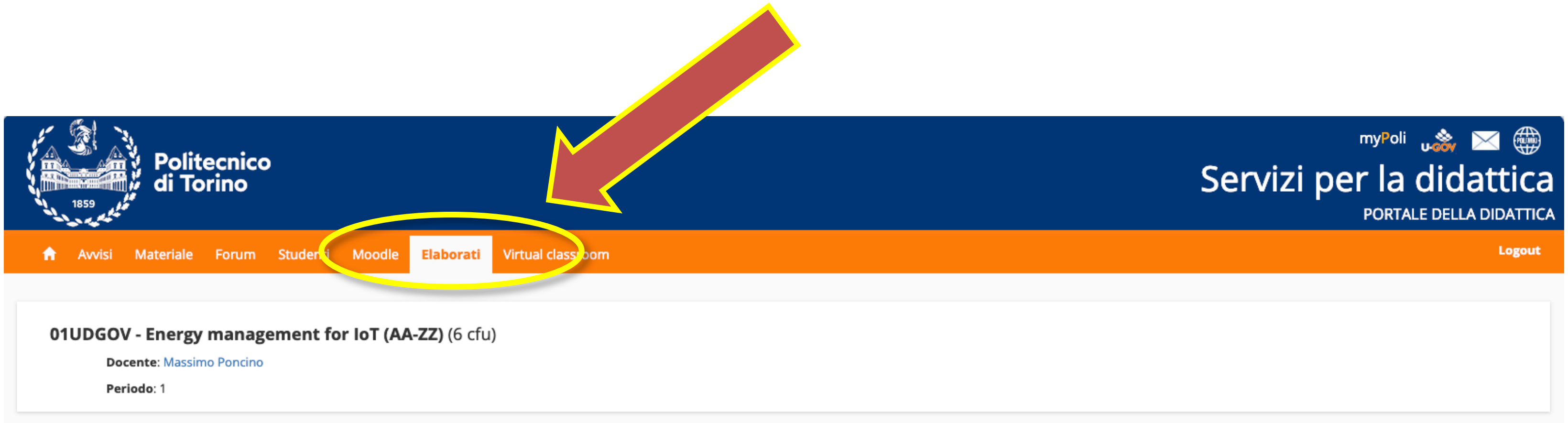
You *MUST* respect this format.

LAB delivery

- What to deliver for each lab:
  - Source code
    - All code you modified/added and consider necessary
    - Do not put files generated by the compiler.
  - Report
    - 5-10 pages per lab, depending on the depth of experiments
    - PDF format
    - **Analysis of results**
      - This is what gives you points!
      - Implementing the code is not enough!

LAB delivery

- How to deliver:
  - Through the didattica web site
    - «Elaborati/Homework» tab



Lab 1
Dynamic Power Management

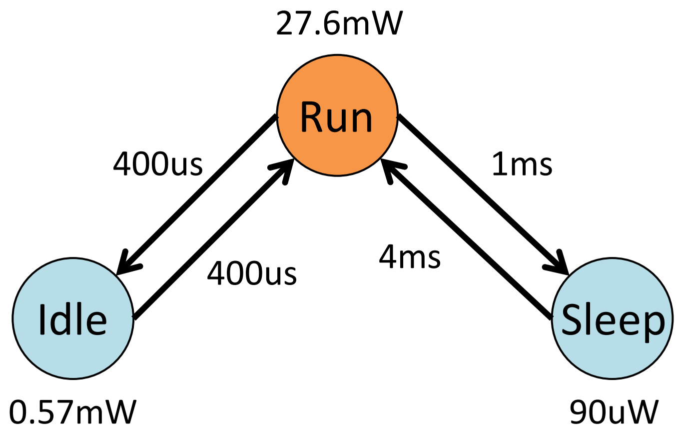
Objective and organization

- Understanding of the basics of DPM
  - Use and modify a simple power state machine simulator in C
  - Evaluation of power management policies

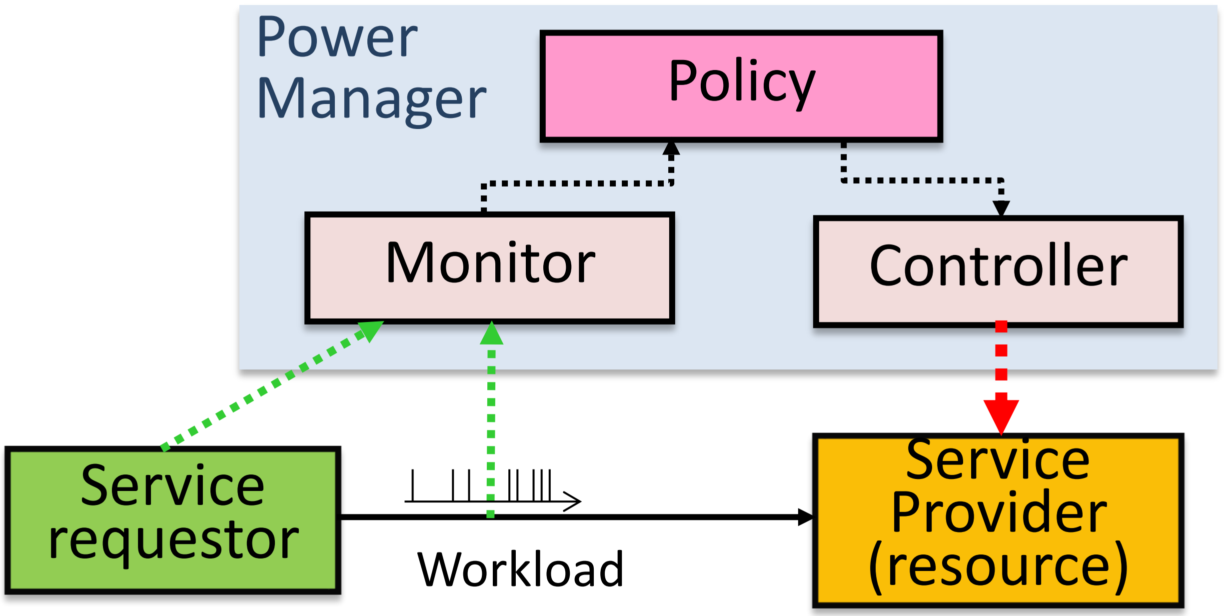
- 1 report and 3 days

Recall

- Dynamic Power Management
  - Reduce energy by turning devices to low power states when peak performance is not needed
  - Devices abstracted as power state machines
    - Several internal states corresponding to modes of operation
      - Different power and service levels



Recall



- Power manager (PM)
  - Monitors requestor's activity and sets state of provider according to some **policy**
    - E.g., shuts down component after some inactivity time

Recall

- **Optimization Problem:** Given a PSM and a workload, determine the optimal allocation of power states over time that minimizes total energy under performance constraints

- Non-idealities make the problem non-trivial!
  - Transitions costs (time & energy) are not zero
    - Transitions must be amortized!
  - Length of inactivity periods often unknown

DPM simulator

- Goal of the lab:
  - Evaluate on a case study how energy saving changes as a function of
    - The applied DPM policies
    - The distribution of inactivity times
    - The PSM parameters
    - …

DPM simulator

- C program with the following basic operations
  - Read a power model → a PSM
  - Read a workload profile
  - Simulate two power management policies
    - Timeout
    - History-based prediction

Code, PSM definition and workloads available on course web page

DPM simulator

- **`dpm_simulator [-help] [-t|-h] [-psm <psm file>][-wl <wl file>]`**

  `-help:` prints command line instructions and returns

  `-t <Timeout>:` use a *timeout* policy with <Timeout>

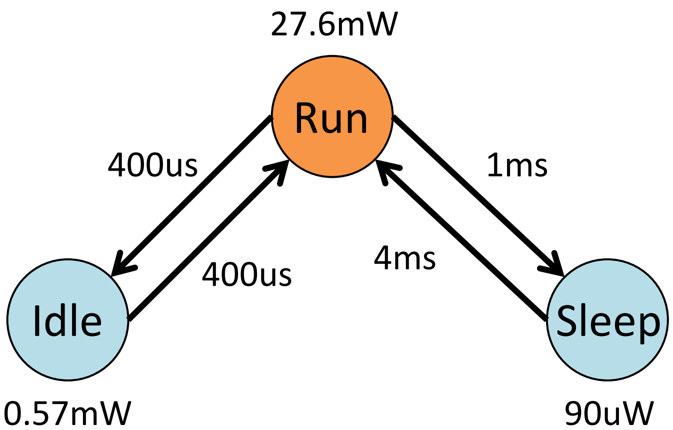
  `-h <Value1> …<ValueN>:` use a *history-based predictive policy.* <Value1-N> are additional policy parameters

  `-psm <psm file>:` the name of the file describing the *power state machine* (PSM) of the resource

  `-wl <wl file>:` the *workload* file name

Format of the PSM

27.6	0.57	0.09	States power
0/0	0.01/0.4	0.02/1	Transitions costs (energy/time)
0.01/0.4	0/0	-1/-1	
2/4	-1/-1	0/0	



Special values (non-existing transitions):

- 0/0: Self-loops (i.e., state does not transition to itself)
- -1/-1: There is no transition between states

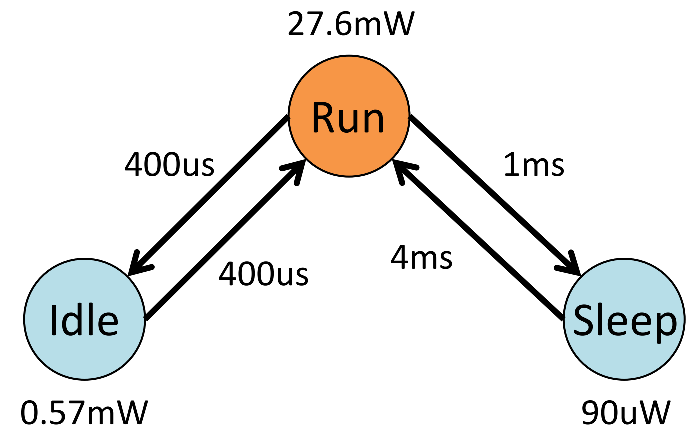
Default time, power, and energy units are: ms, mW and mJ

Format of the PSM

27.6	0.57	0.09
0/0	0.01/0.4	0.02/1
0.01/0.4	0/0	-1/-1
2/4	-1/-1	0/0

States power

Transitions costs (energy/time)



TO STATE...

FROM STATE...

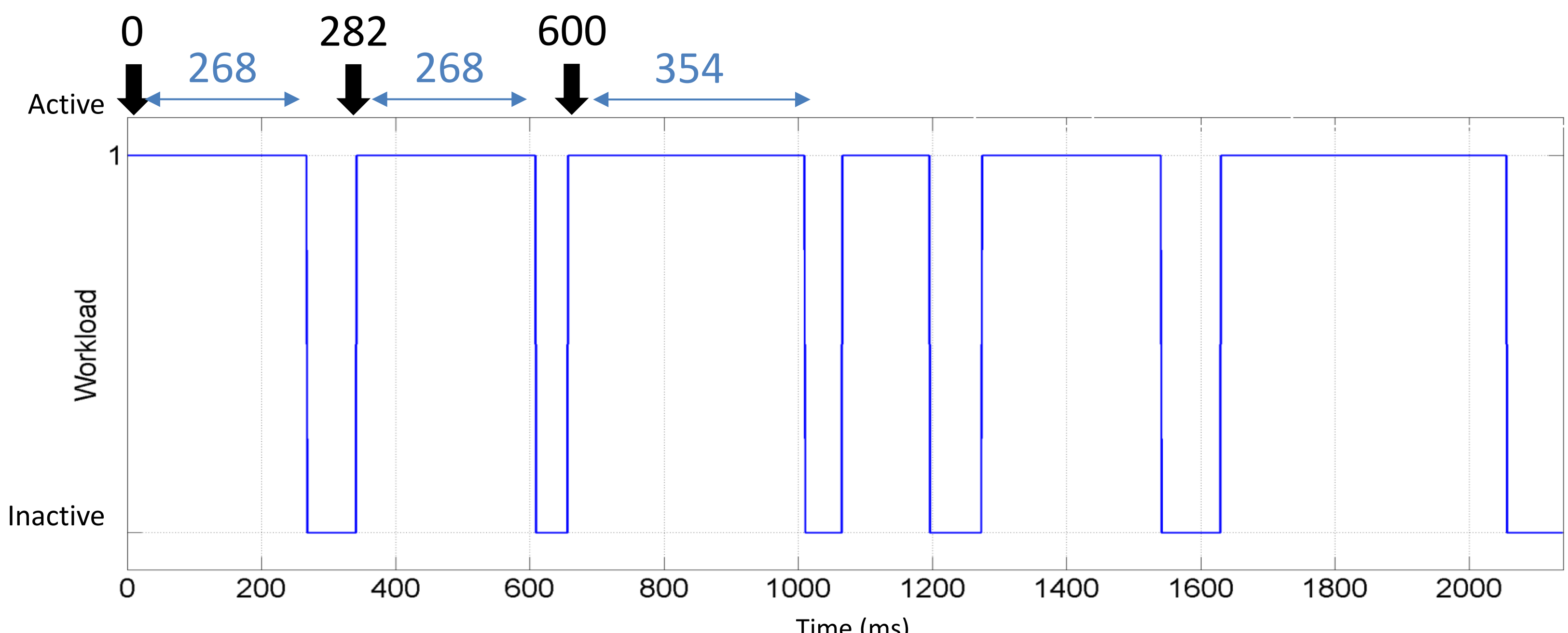
	RUN	IDLE	SLEEP
RUN			
IDLE			
SLEEP	2 / 4		

ENERGY / TIME
2 mJ / 4ms

Workload format

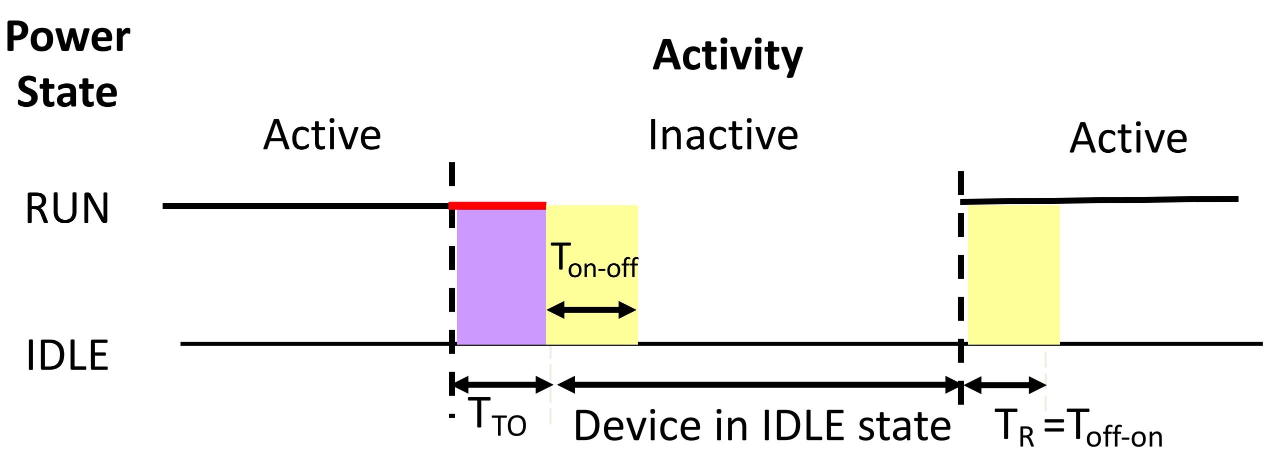
- The workload is given as a list of work items requested to the resource by the service requestor
  - Values are in **ms**

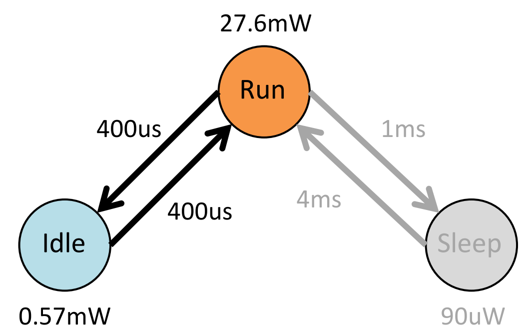
Arrival	Duration	
0	268	1st Work Item
282	268	2nd Work Item
600	354	3rd Work Item
…		…



DPM policies

- Timeout policy
  - Observe the first part of the current inactive period to predict the length of the remaining part
  - Put the device in IDLE state T_{TO} time units after it has become inactive





Compile and execute

- Compile (requires gcc):

  make

- Generate Documentation (requires doxygen):

  make docs

  - Generates «docs» folder with HTML documentation

- Execute:

  ./dpm_simulator -t 20 -psm example/psm.txt –wl example/wl.txt

  - Timeout policy with timeout value 20ms

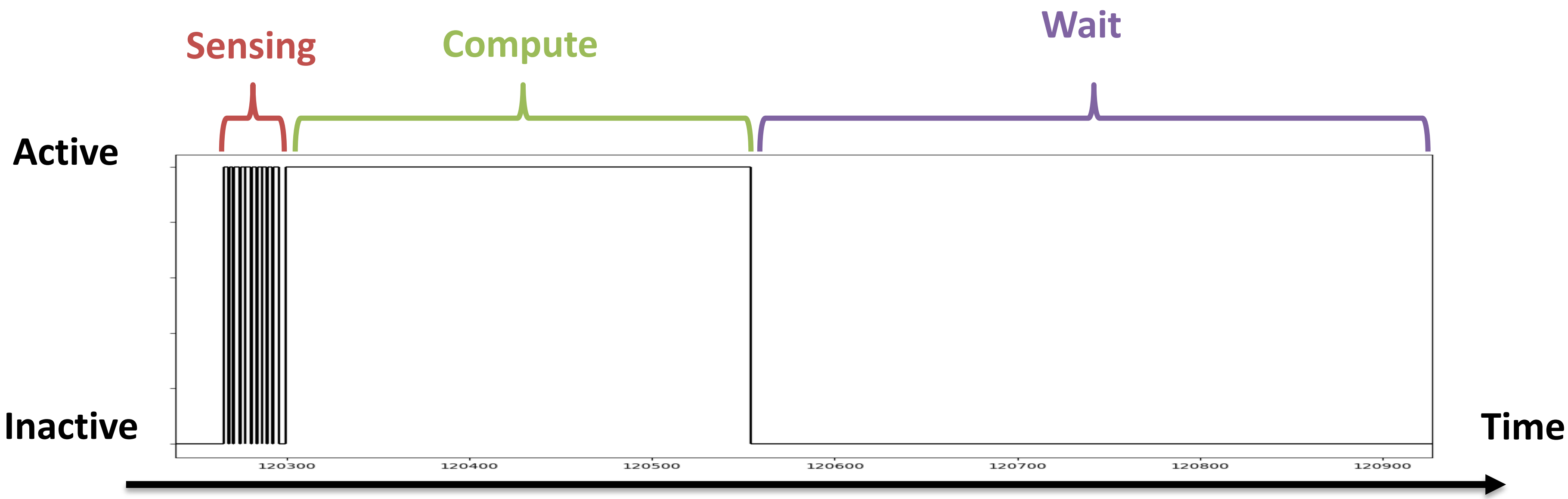
Compile and execute

```
→  dpm-simulator ./dpm_simulator -psm example/psm.txt -wl example/wl.txt -t 20
[psm] State Run: power = 27.6000mW
[psm] State Idle: power = 0.5700mW
[psm] State Sleep: power = 0.0900mW
[psm] Run -> Idle transition: energy  = 0.0100mJ, time = 0.4000ms
[psm] Run -> Sleep transition: energy  = 0.0200mJ, time = 1.0000ms
[psm] Idle -> Run transition: energy  = 0.0100mJ, time = 0.4000ms
[psm] Sleep -> Run transition: energy  = 2.0000mJ, time = 4.0000ms

[sim] Active time in profile = 300.130000s
[sim] Inactive time in profile = 243.921000s
[sim] Tot. Time w/o DPM = 544.051000s, Tot. Time w DPM = 544.051500s
[sim] Total time in state Run = 323.694200s
[sim] Total time in state Idle = 219.444500s
[sim] Total time in state Sleep = 0.000000s
[sim] Timeout waiting time = 23.564200s
[sim] Transitions time = 0.912800s
[sim] N. of transitions = 2282
[sim] Energy for transitions = 0.0228200000J
[sim] Tot. Energy w/o DPM = 15.0158076000J, Tot. Energy w DPM = 9.0818632852J
→  dpm-simulator
```

Workloads

1. Two workloads provided on the course page
   - Similar structure (typical IoT workloads)
     - Read some data from sensors (waiting)
     - Elaborate the data (e.g., Neural Network inference)
     - Repeat every 2 minutes

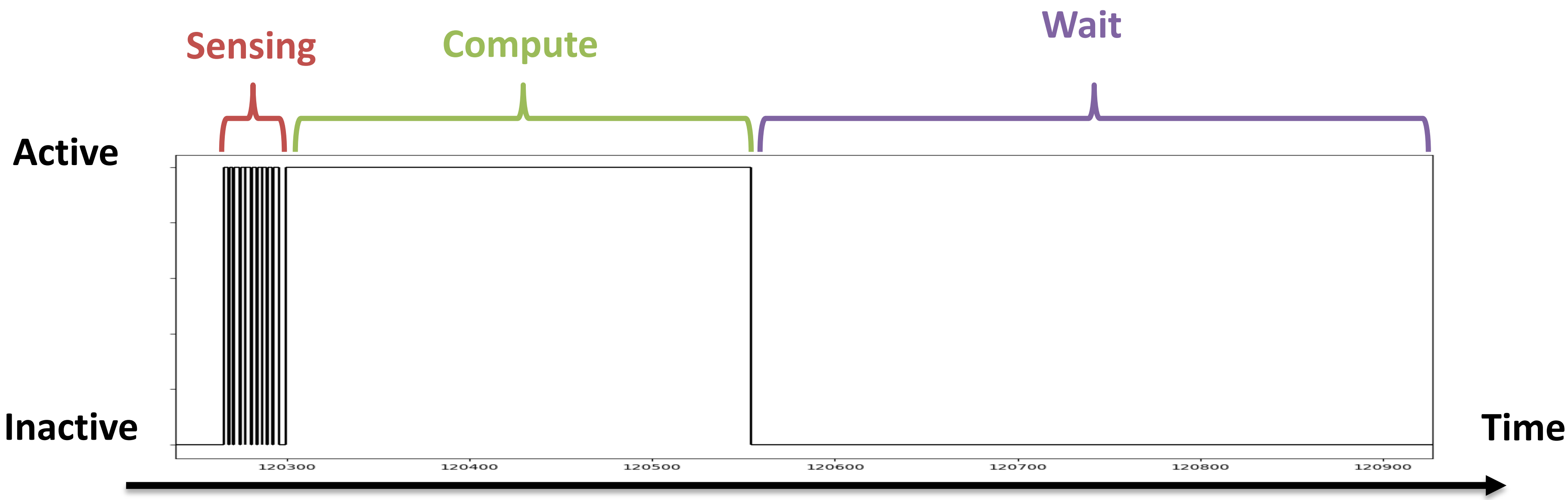


Workloads

1. Two workloads provided on the course page
   - Workloads generated using an embedded system simulator called JUMPER.
     - You **don't have** to use JUMPER directly for the lab, but you can (if you want) use it to generate additional workloads
     - Instructions and a basic simulation setup are provided on the course page.
       - Note that the "workload" generated by the provided Jumper code uses a different format (INACTIVE_START, INACTIVE_END) instead of (ACTIVE_START, ACTIVE_DURATION) but the conversion should be straight-forward.

Workloads

1. Two workloads provided on the course page
   - workload_1.txt: ”fast” sensors → 4ms to return a value
   - workload_2.txt: “slow” sensors → 100ms to return a value
   - **Does this have an impact on DPM?** It’s your job to find it out



Part 1
Default Timeout Policy

Assignment 1 – Part 1

1. Compile the DPM simulator
2. Test it with the two workloads, using the default timeout policy
   – Only transitions between RUN and IDLE states
   – Try different timeout values and see what happens. Discuss in your report **why** it happens!

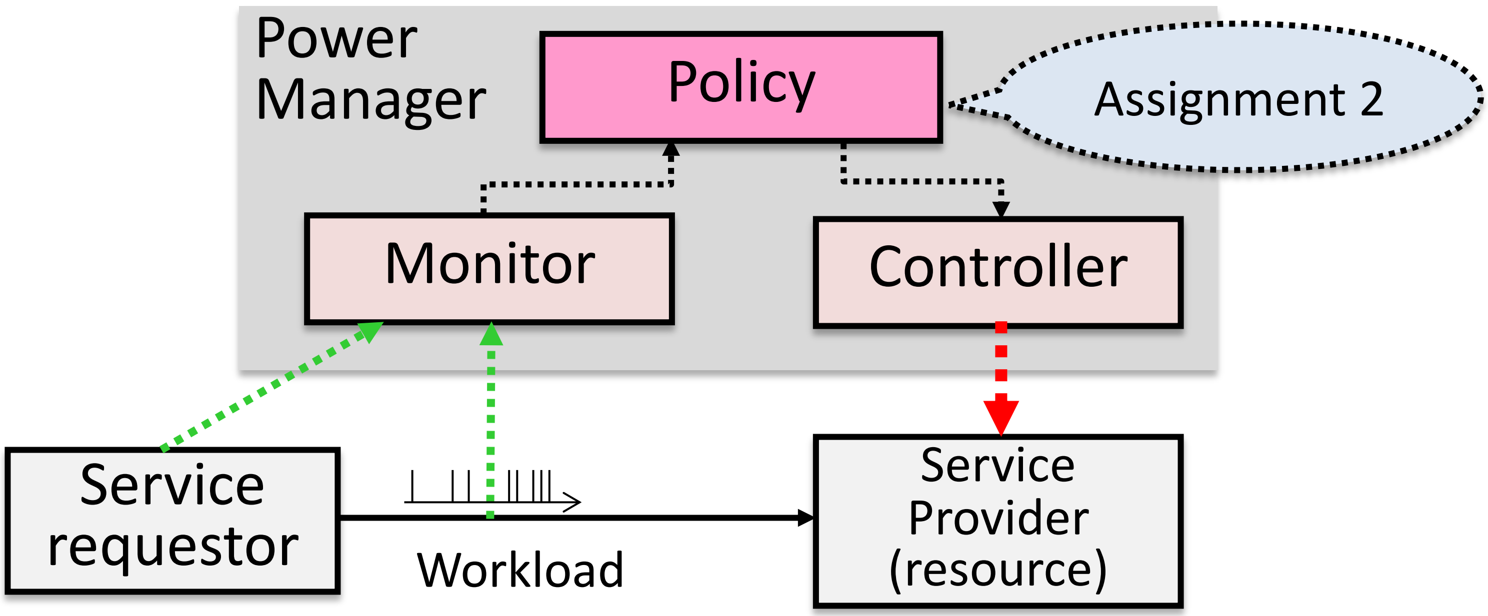
Part 2
Extension of the timeout policy

DPM simulator

- The DPM simulator supports a timeout policy with transitions from RUN to IDLE
  - Never goes to SLEEP state
  - Moving to SLEEP may save even more energy…

[sim] Active time in profile = 300.130000s
[sim] Inactive time in profile = 244.066000s
[sim] Total time = 544.196000s
[sim] Timeout waiting time = 24.679000s
[sim] Total time in state Run = 324.809000s
[sim] Total time in state Idle = 219.387000s
[sim] Total time in state Sleep = 0.000000s
[sim] Time overhead for transition = 0.910800s
[sim] N. of transitions = 2277
[sim] Energy for transitions = 0.0227700000J
[sim] Energy w/o DPM = 15.0198095999J, Energy w DPM = 9.1125489900J

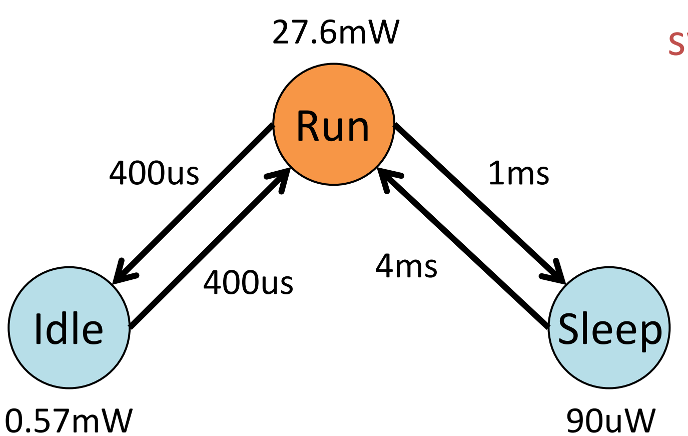
Recall



- Policy implementation
  - Modify the timeout-based policy

Assignment 1 – Part 2

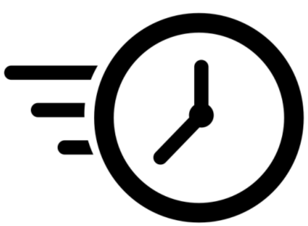
- Modify the timeout policy to enable transitions also to SLEEP
  - Must modify the implementation of the DPM simulator




```
switch (policy) {

  case DPM_TIMEOUT:
    /* Day 2: EDIT */
    if(curr_time > t_inactive_start + tparams.timeout) {
      *next_state = PSM_STATE_IDLE;
    } else {
      *next_state = PSM_STATE_RUN;
    }

    break;
```



The DPM simulator

- `print_command_line()` (src/utilities.c)
  - Prints the command line to invoke the tool
- `parse_arg()` (src/utilities.c)
  - Parses the inputs you provide via command line
  - For the timeout policy:

    ```
    *selected_policy = DPM_TIMEOUT;
    tparams->timeout = atof(argv[++cur]);
    ```

    value passed as parameter after -t
  - May have to be modified, too

The DPM simulator

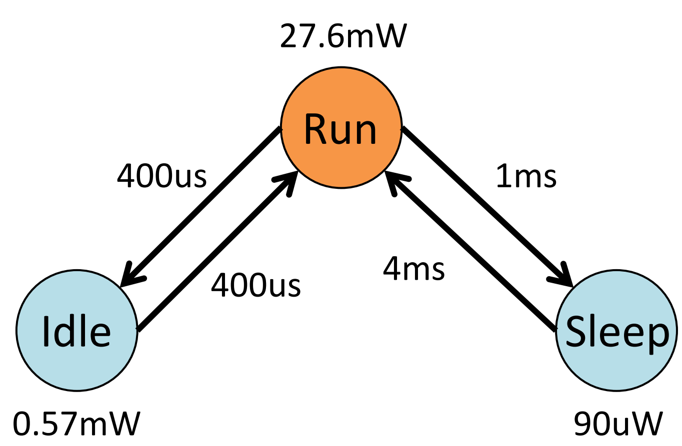
- **`dpm_simulate()`** (src/dpm_policies.c)
  - Emulates policy application to the PSM given the workload
    - Invokes **`dpm_decide_state()`** repeatedly, with a given time granularity, during inactivity periods, to apply the policy and determine state transitions
    - Computes time and energy statistics for the simulation
    - Compares with a no-DPM approach.
    - The code is quite straight-forward, have a look for yourself…

The DPM simulator

- `dpm_decide_state()` (src/dpm_policies.c)
  - Determines the next Power State of the system **during inactivity times**, based on:
    - Current simulation time
    - Time in which the system became inactive
    - Previous state (not used and not necessarily needed)
    - History of previous inactivity periods (Used in Part 3)
    - Adopted policy (e.g., timeout vs predictive) and its parameters (e.g., T_{TO})

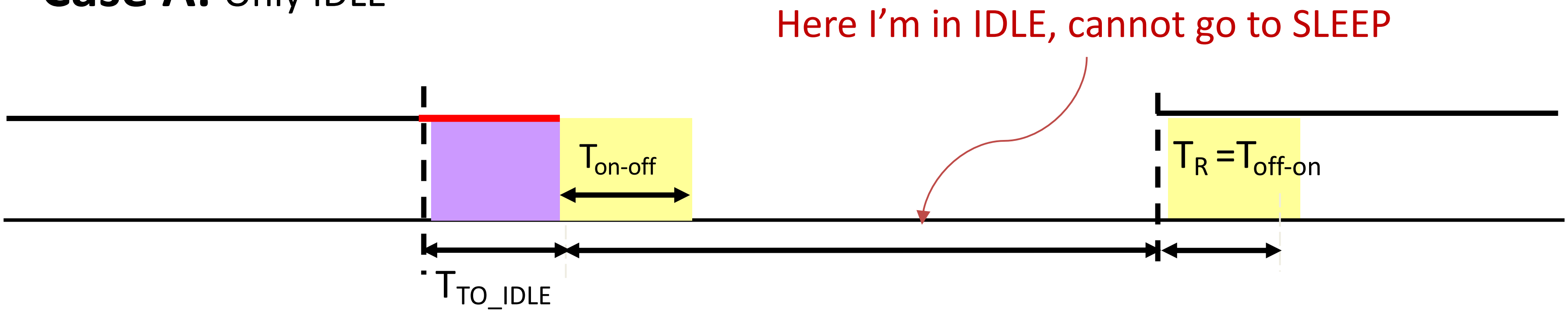
Power State Machine

- Note that there is **no transition** between IDLE and SLEEP in the PSM
  - This system can only go to IDLE **or** SLEEP **from the RUN state.**
  - With a timeout policy **only one of the two low-power states can be selected**
    - Different story if we use a predictive policy.

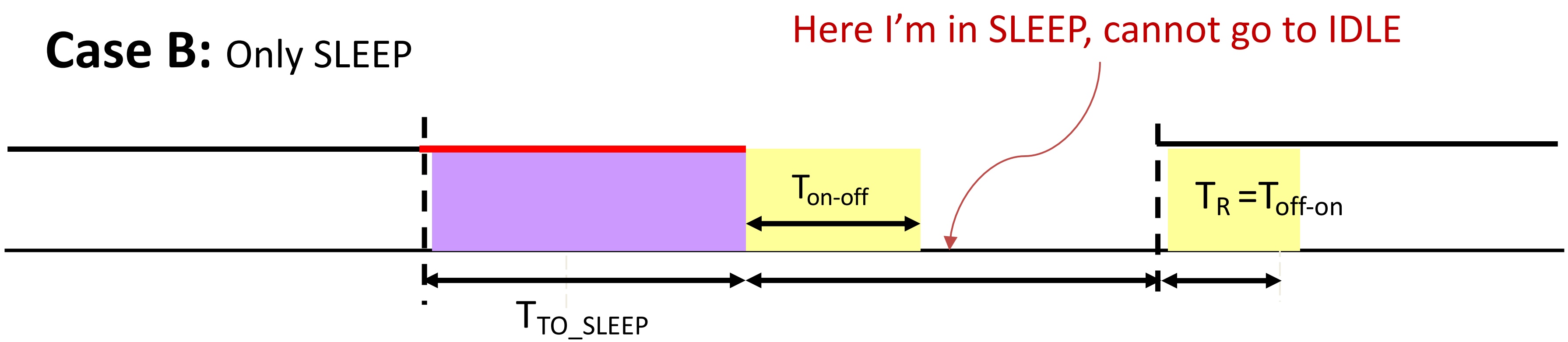


Timeout Policy

Case A: Only IDLE



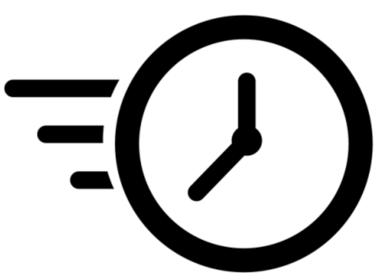
Case B: Only SLEEP



NOTE: There is actually a way to use both low-power states. Which one? [Possible Extra]

Assignment 1 – Part 2

- Report assignment:
  - Comparison between RUN-> IDLE timeout policy and RUN->SLEEP timeout policy
    - What changes?
    - What's the **best** $\mathbf{T_{TO}}$ value in the two cases?
    - Which of the two results in the overall lower energy? What changes for the two **workloads** provided?
    - **Why?**



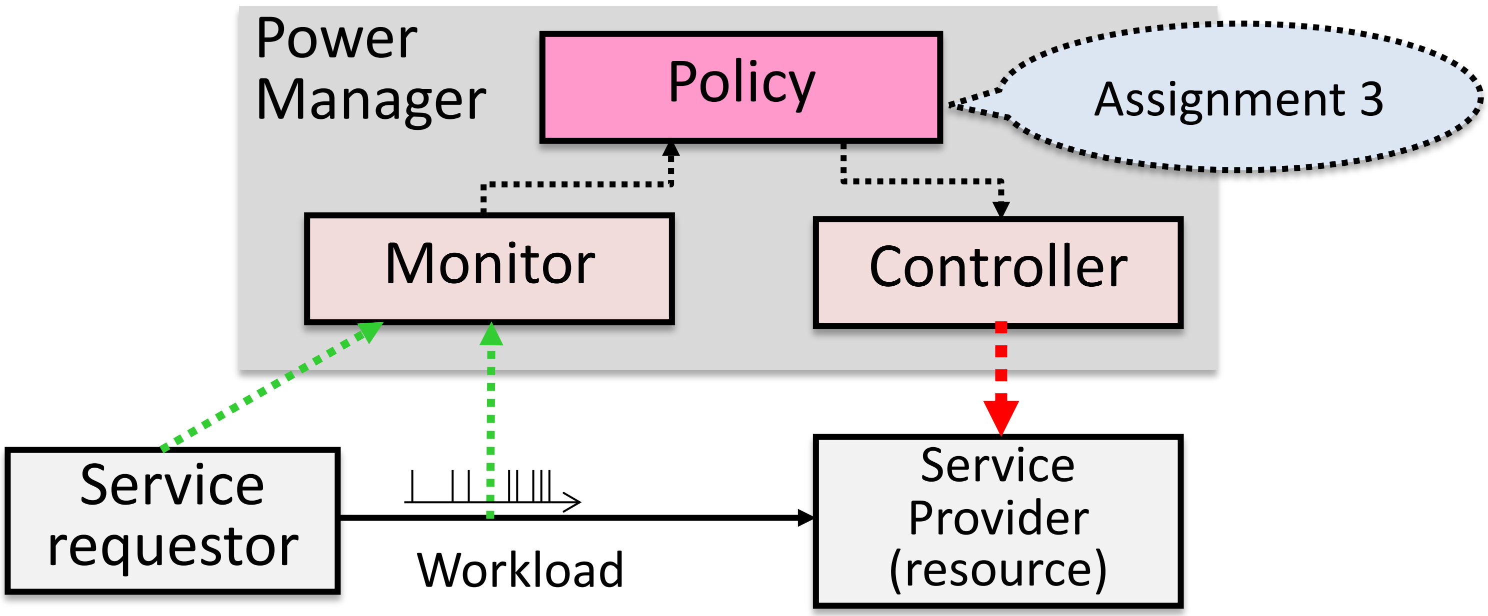
This is the most important!!!

Assignment 1 – Part 2

- **Extra:** Make your analysis **automatic** and **systematic**
  - Don't just try some "random" T_{TO} values
  - Compare things in a reasonable and meaningful way

Part 3
Predictive Policy

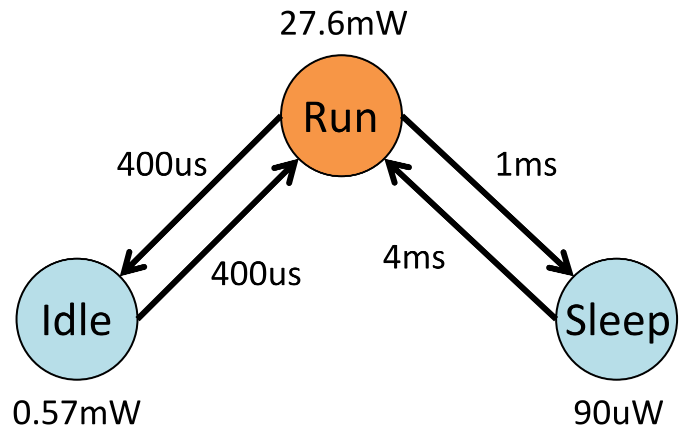
Recall

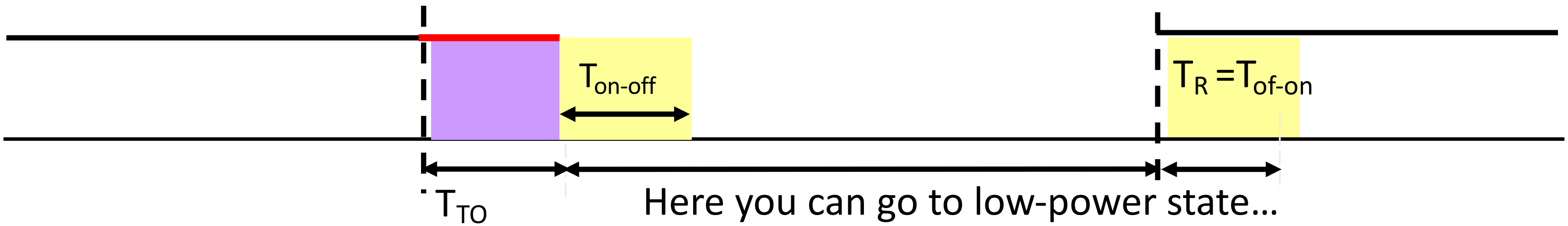


- Policy implementation
  - Implement a predictive policy

Recall

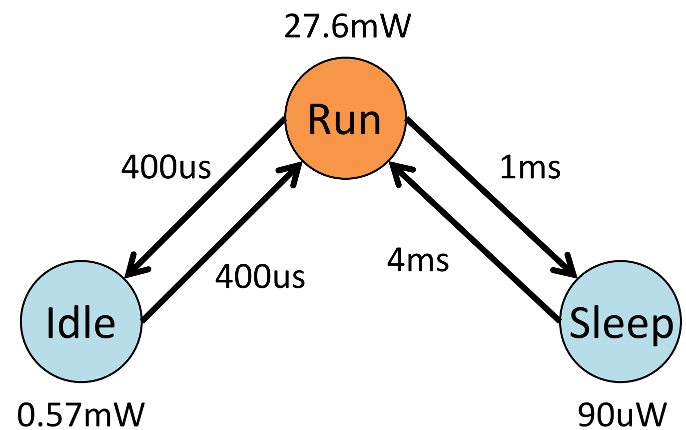
- So far, we worked with timeout policies…
  - Put the device in off state T_{TO} time units after it has become inactive



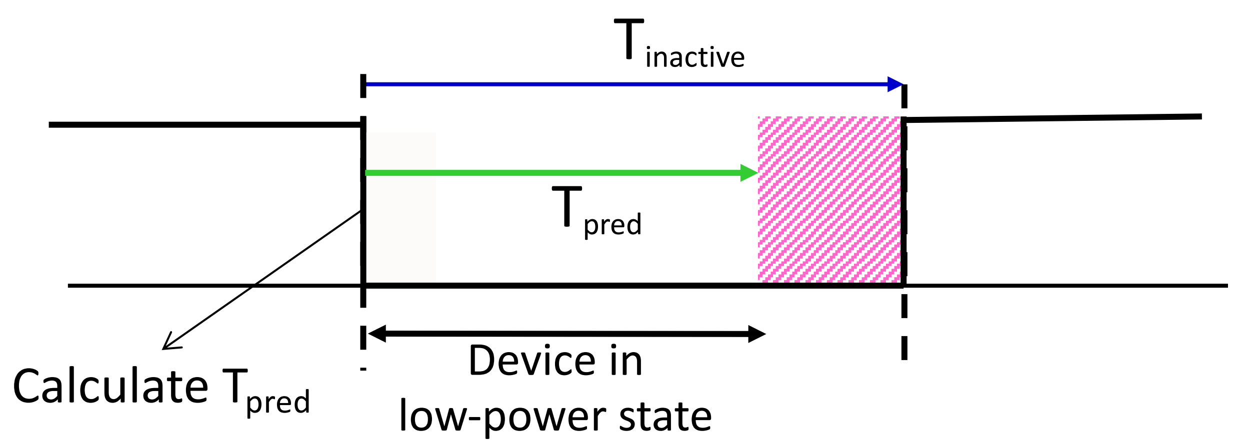


Recall

- So far, we worked with timeout policies…
  – Put the device in off state T_{TO} time units after it has entered the idle state

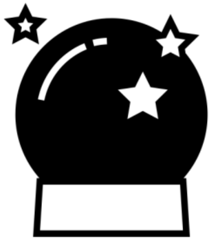


- Can we do better?



Recall

- Predictive policies
  - Predict inactive period T_{pred} ~ $T_{inactive}$
  - Go to low-power state if T_{pred} is long enough to amortize state transition cost

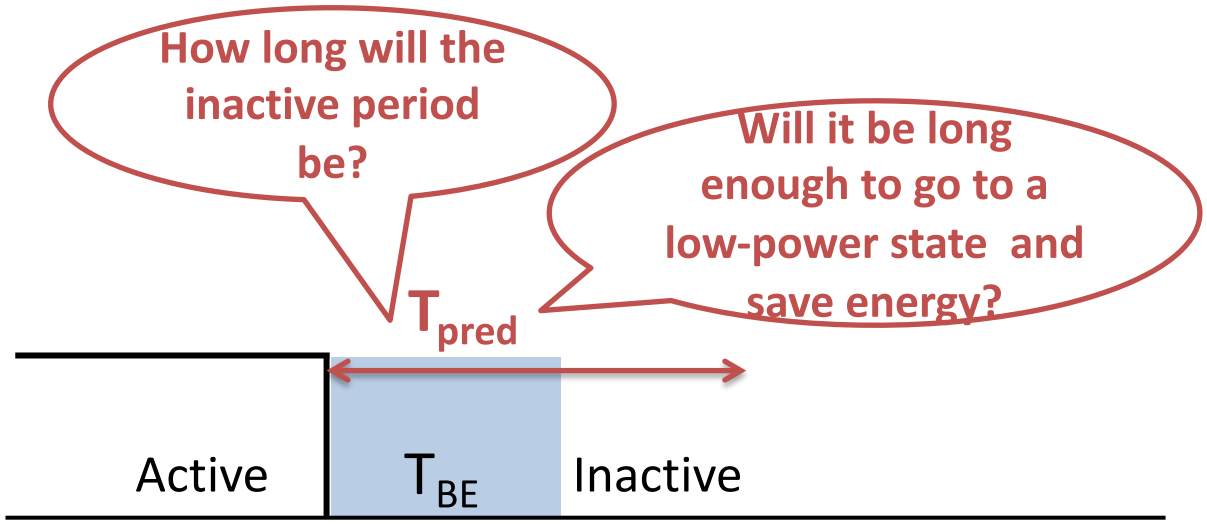


- History-based policies:
  - Predictive policies that use previous **history** of **T_{active} and/or $T_{inactive}$**
  - Example: regression equation

$$T_{pred}[i] = \boldsymbol{K} + \boldsymbol{K_1} \cdot T_{inactive}[i-1] + \boldsymbol{K_2} \cdot T_{inactive}[i-2] + \boldsymbol{K_3} \cdot T_{inactive}[i-3]$$

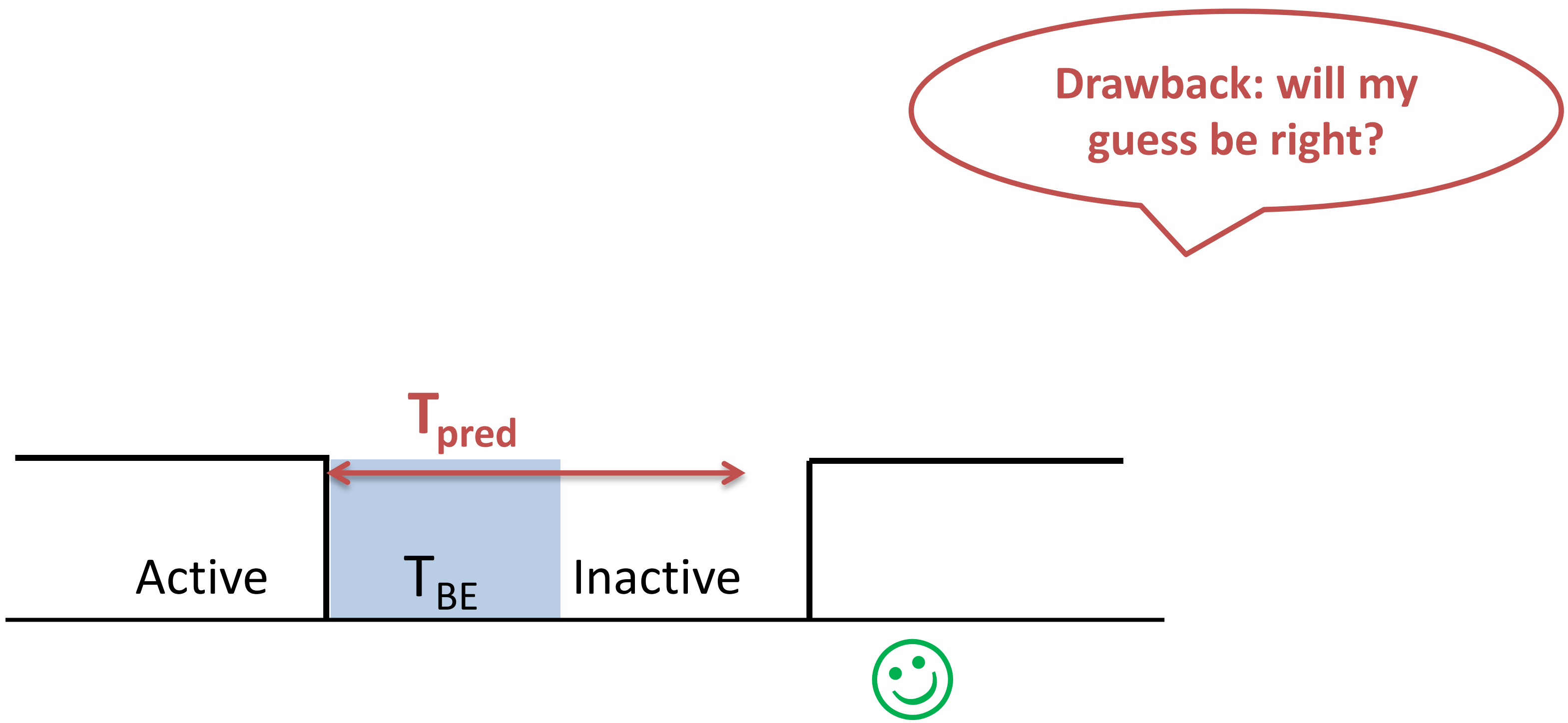
Recall

- Goal of predictive policies



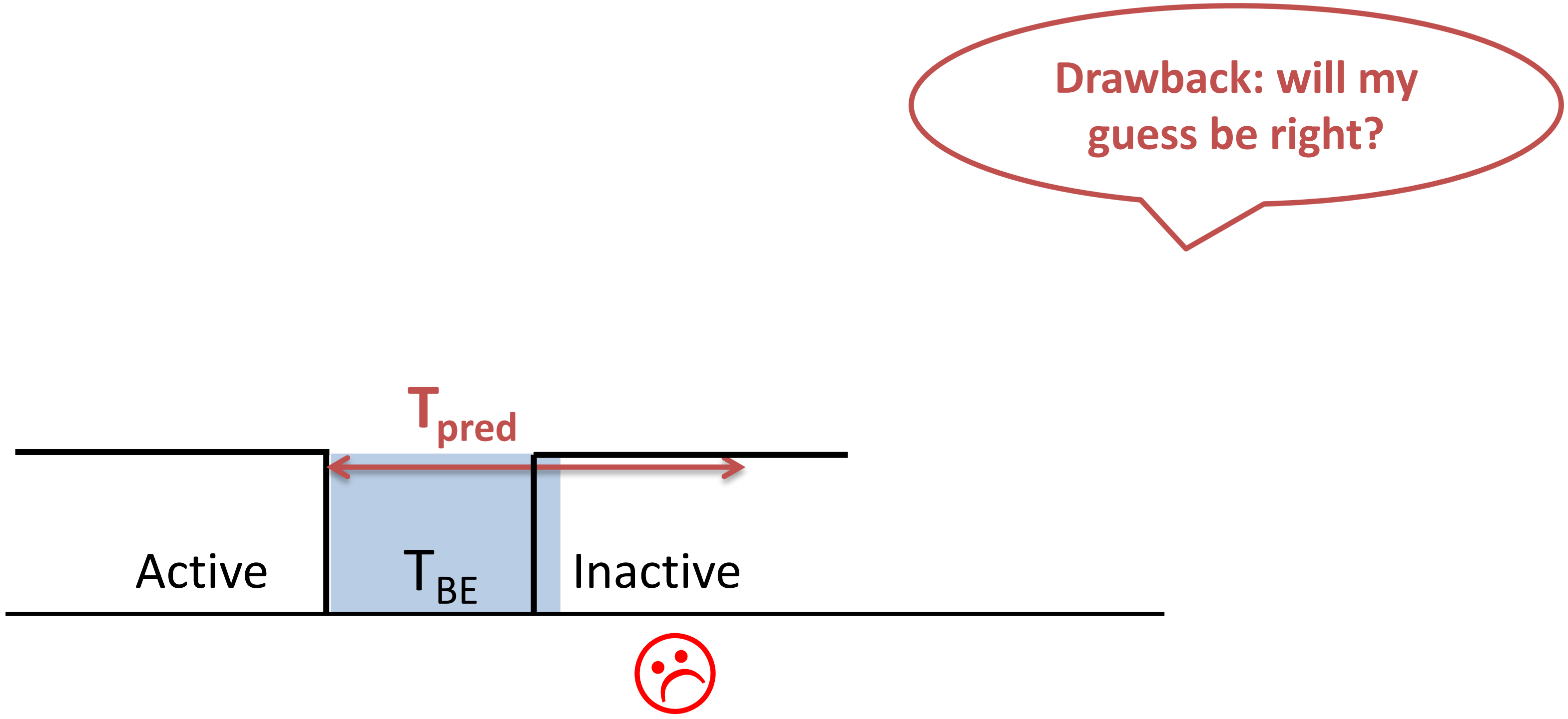
Recall

- Drawback of predictive policies
  - Under-prediction



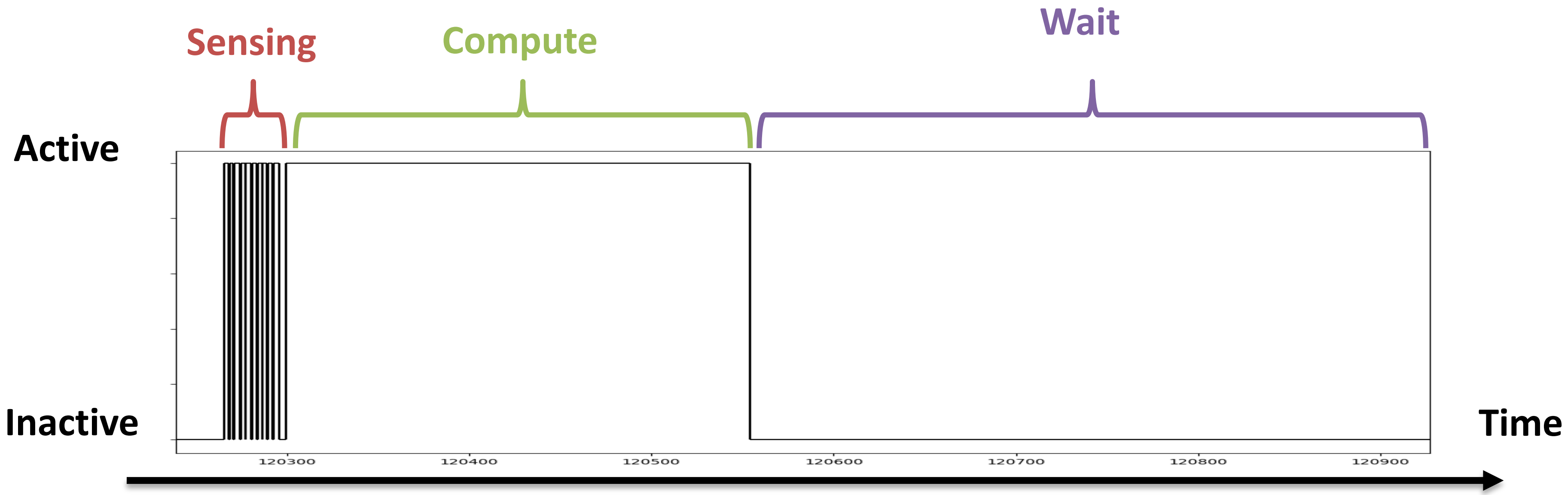
Recall

- Drawback of predictive policies
  - Over-prediction



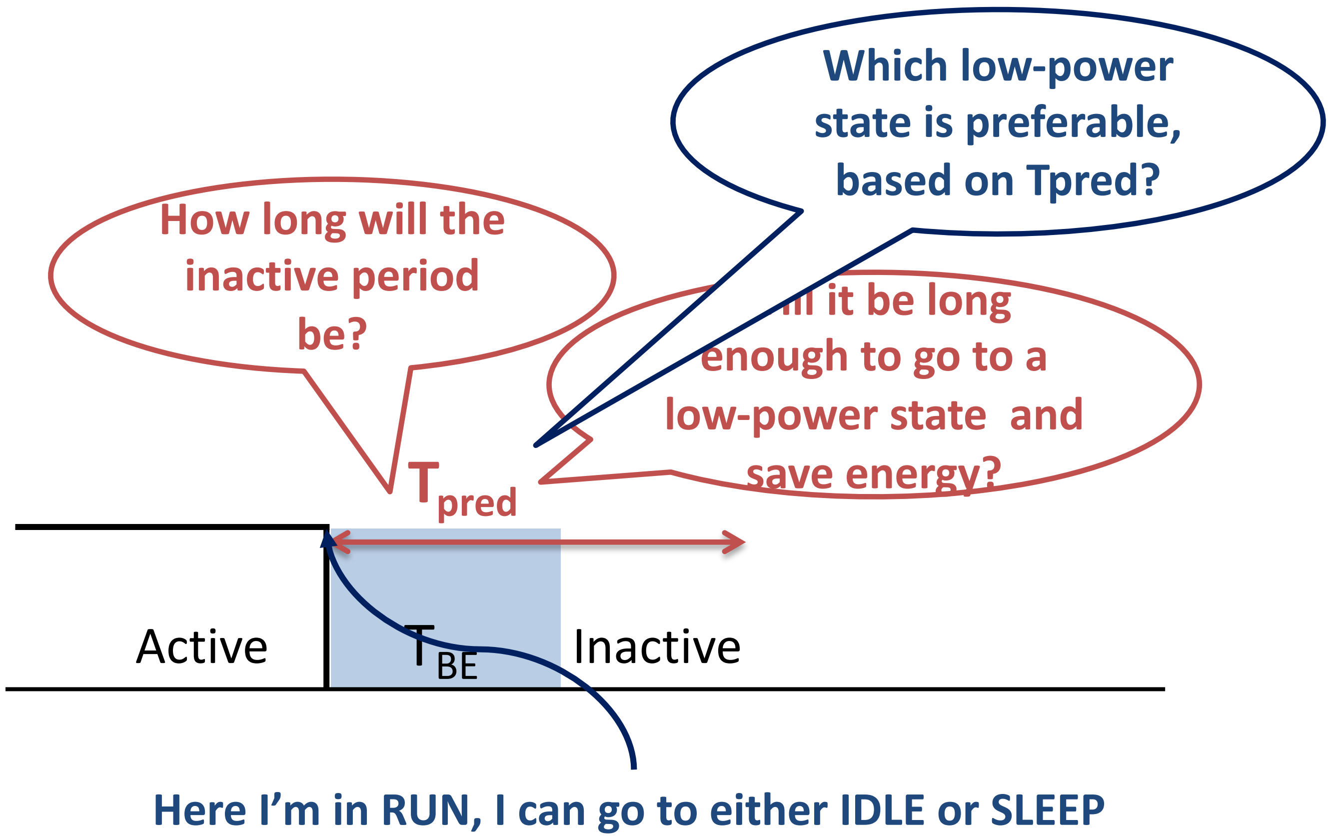
Recall

- But remember our workloads:
  - Pretty easy to predict (roughly) the duration of the next inactive phase.
  - How?



Recall

- A predictive policy also allows us to use both low-power states in the PSM:



The DPM Simulator

- Currently the simulator contains data structures and functions to implement a **regression-based** policy:
  - Something like:

$$T_{pred}[i] = \boldsymbol{K} + \boldsymbol{K_1} \cdot T_{inactive}[i-1] + \boldsymbol{K_2} \cdot T_{inactive}[i-2] + \boldsymbol{K_3} \cdot T_{inactive}[i-3]$$

  - *Mainly for «historical reasons»*
  - **May not be the best type** of history-based policy to implement

The DPM Simulator

To be modified (probably)

```
int parse_args(…) {

    //…
    if(strcmp(argv[cur], "-h") == 0) {
        *selected_policy = DPM_HISTORY;
        if(argc > cur + DPM_HIST_WIND_SIZE + 2){
            int i;
            for(i = 0; i < DPM_HIST_WIND_SIZE; i++) {
                hparams->alpha[i] = atof(argv[++cur]);
            }
            hparams->threshold[0] = atof(argv[++cur]);
            hparams->threshold[1] = atof(argv[++cur]);
        } else return 0;
    }
```

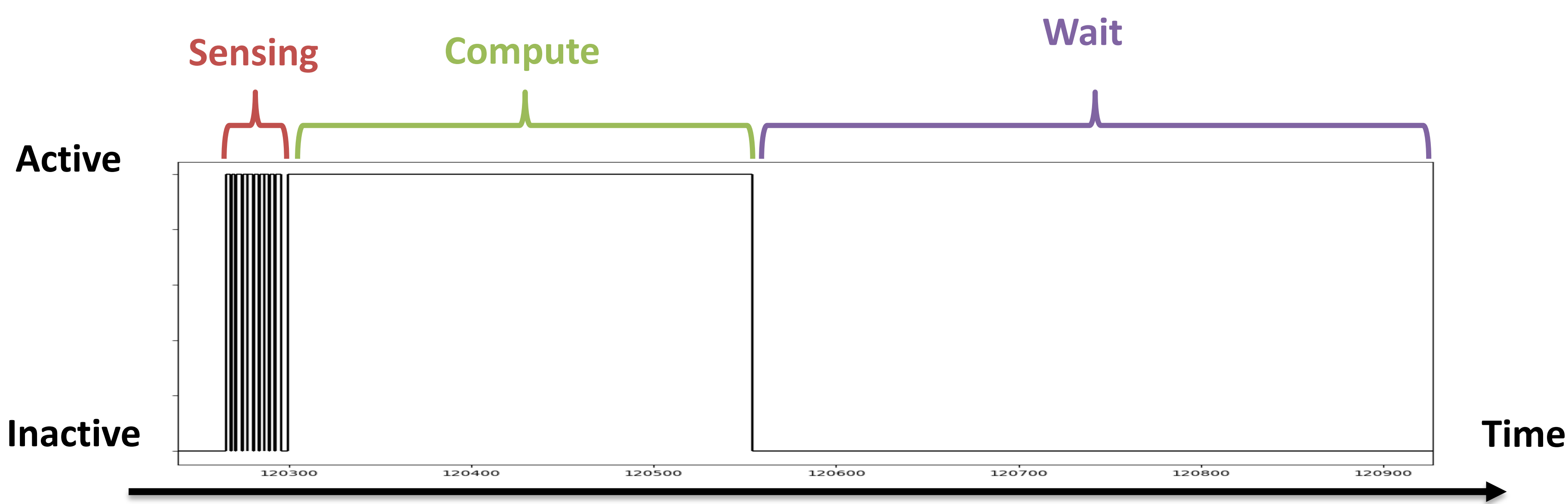
The DPM Simulator

To be modified

```
/* update inactive time history */
void dpm_update_history(…) {
        for (int i=0; i<DPM_HIST_WIND_SIZE-1; i++){
                h[i] = h[i+1];
        }
        h[DPM_HIST_WIND_SIZE-1] = new_inactive;
}
```

The DPM Simulator

- Do we really need a polynomial?
- …or can we make our decision simpler?
  - What determines if the next inactive period will be long or short?

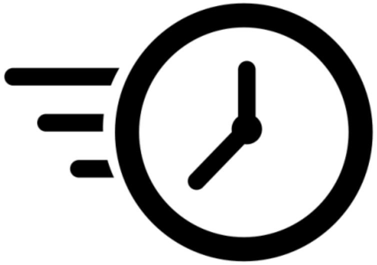


Assignment 1 – Part 3

- Modify the simulator to implement a *predictive policy*
  - It can be **of any kind**, you decide
  - Trying and comparing more than one policy is also good
  - Of course, motivate your choices.

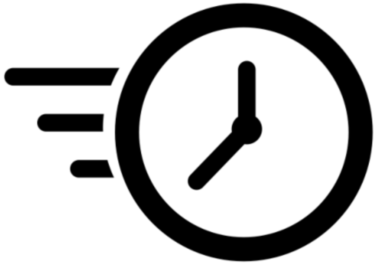
```
case DPM_HISTORY:
        /* Day 3: EDIT */
        *next_state = PSM_STATE_RUN;
        break;
```

To be modified



Assignment 1 – Part 3

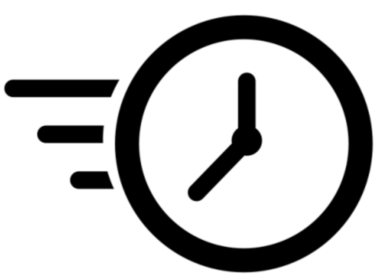
- Report assignment
  - Description of implemented predictive policy
  - Result of implemented predictive policy with the workload profiles
    - Analysis on effect of policy parameters (if any)



Assignment 1 – Part 3

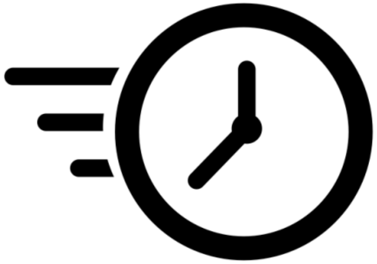
- Report assignment:
  - Comparison between predictive policy and timeout policies
    - What changes?
    - Which approach is the best for the two workloads?
    - Why does a predictive approach work better/worse on one workload than the other (if it does)?
    - **Why?**

This is the most important!!!



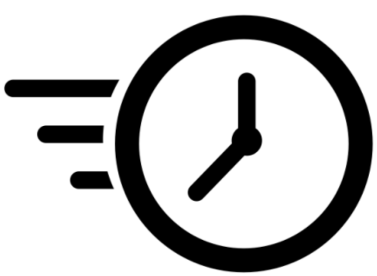
Assignment 1 – Part 3

- **Extras:**
  - Any extra analysis/experiment is well appreciated
    - Show your creativity and desire to explore the topic
  - Extras are **not** mandatory, but can give additional points:
    - If you only do the mandatory points of the previous slides, perfectly, you’ll still get max score



Assignment 1 – Part 3

- **Extras: some suggestions**
  - Change the PSM (i.e., the target system)
  - Change the workload:
    - Generate new “synthetic” workloads using any tool of your choice (MATLAB, Python, etc).
      - You can simply generate the sequence of work items, but they should make sense.
    - Install and use **Jumper** to generate workloads starting from real embedded code.
  - Try other policies…
  - Etc.



End of Lab 1!
Now you're ready to prepare the first report…

The deadline is 23:59 of the day before the 2nd exam