CPU and DRAM monitoring for Zeusd

Wonbin Jin

Goal of Zeus Daemon

CPU energy measurement requires root privileges,
GPU energy optimization requires `SYS_ADMIN` privileges

Don’t want to run application with too much privilege

Provide daemon process that runs with admin privileges and
exposes the minimal set of APIs needed

Running Average Power Limit (RAPL)

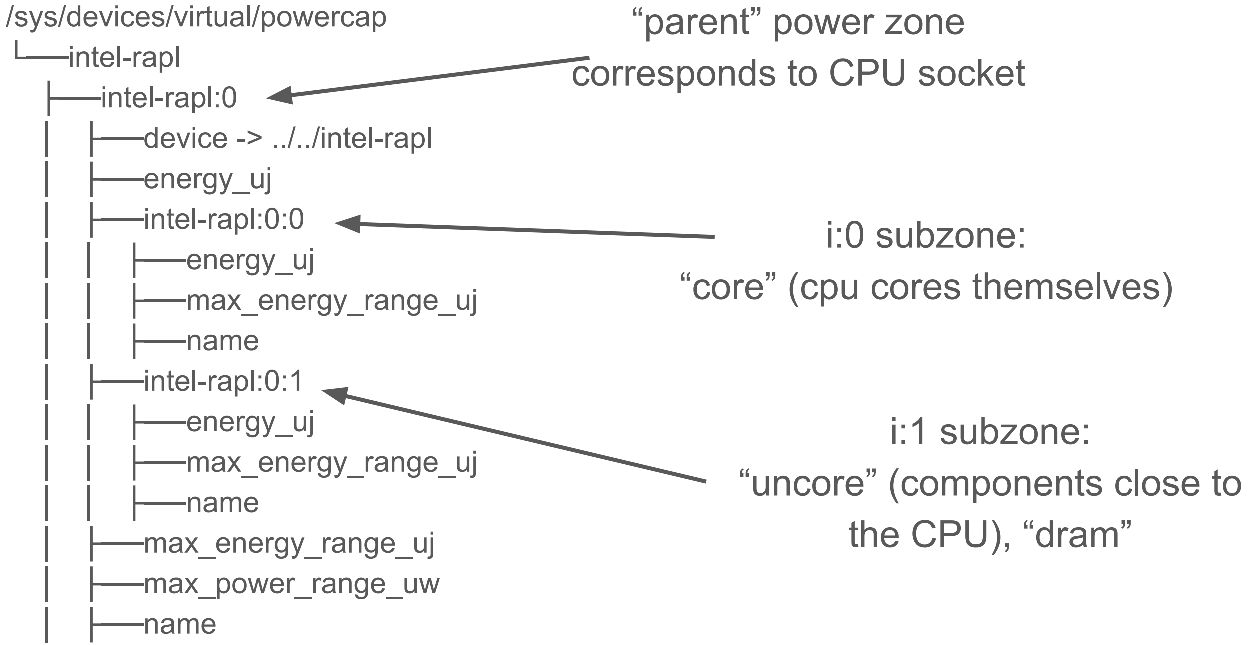
- Feature on Intel processors for real time measurements and power capping of CPU and DRAM

- Supported by most Intel processors and some AMD processors

- Accessed through underlying MSRs, Linux provides an interface through `sysfs`

Running Average Power Limit (RAPL)

- Processor is split into hierarchical “power zones”, each power zone has a subzone



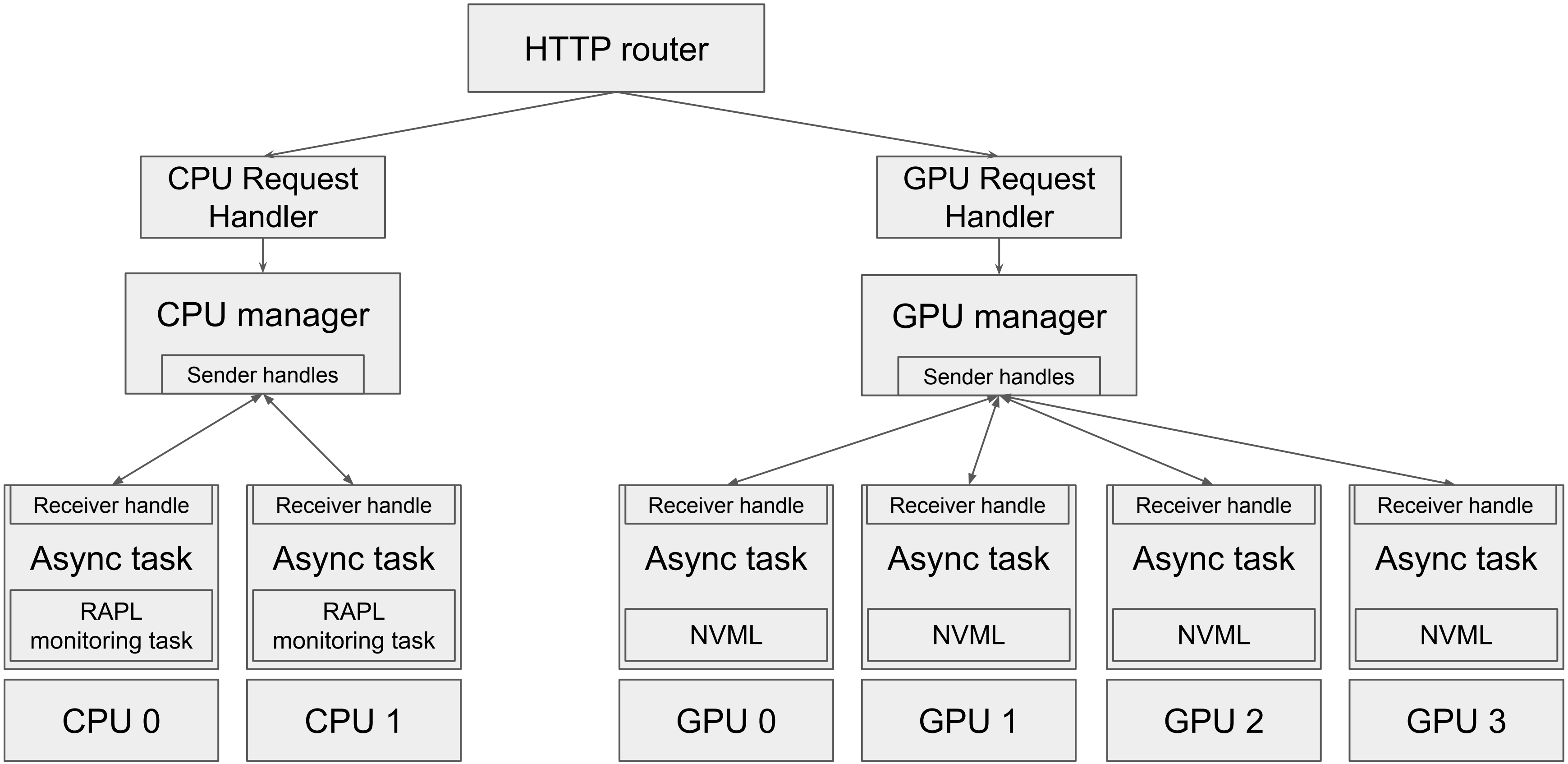
```
/sys/devices/virtual/powercap
└──intel-rapl
    ├──intel-rapl:0
    │   ├──device -> ../../intel-rapl
    │   ├──energy_uj
    │   ├──intel-rapl:0:0
    │   │   ├──energy_uj
    │   │   ├──max_energy_range_uj
    │   │   ├──name
    │   ├──intel-rapl:0:1
    │   │   ├──energy_uj
    │   │   ├──max_energy_range_uj
    │   │   ├──name
    │   ├──max_energy_range_uj
    │   ├──max_power_range_uw
    │   ├──name
```

“parent” power zone corresponds to CPU socket

i:0 subzone:
“core” (cpu cores themselves)

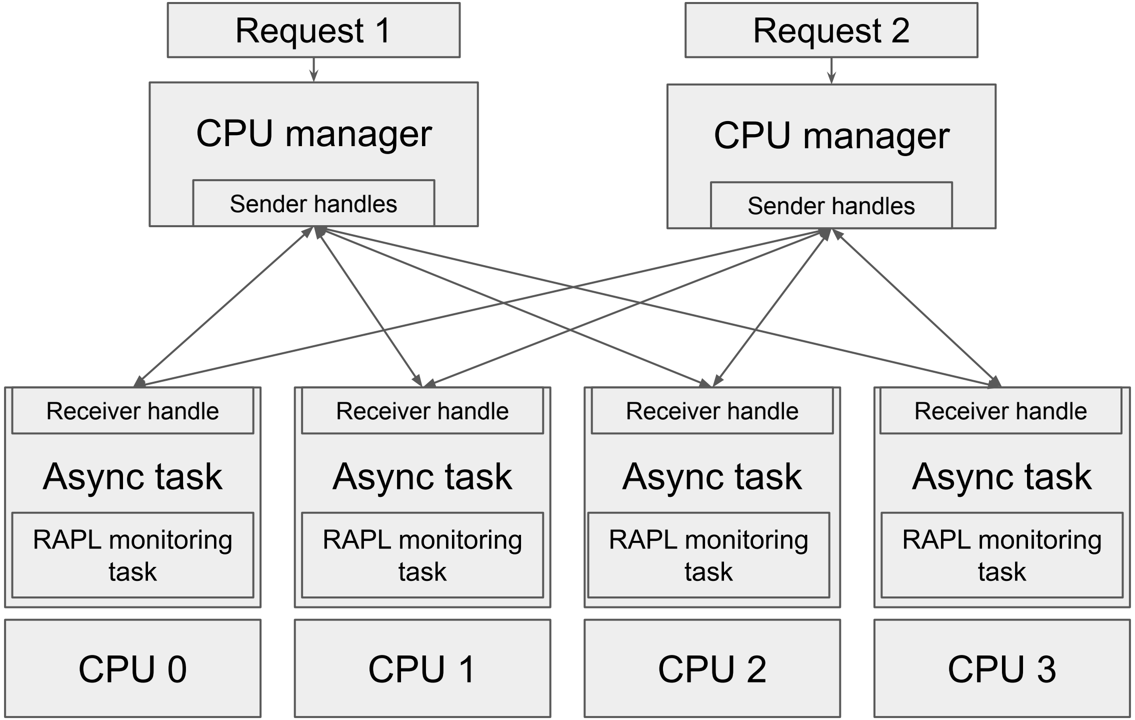
i:1 subzone:
“uncore” (components close to the CPU), “dram”

Zeus daemon (zeusd) – New architecture

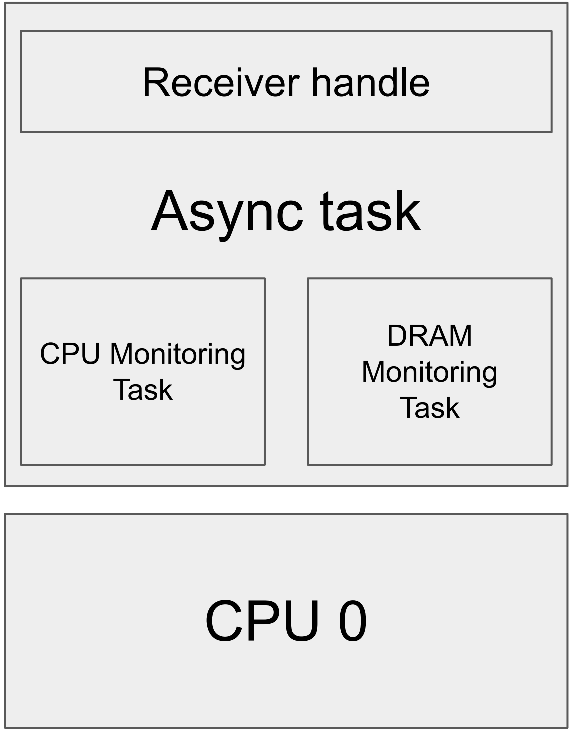


Zeus daemon (`zeusd`) – CPU Monitoring

CPU manager and all senders are **cloned** on each request (i.e., not a singleton/bottleneck)



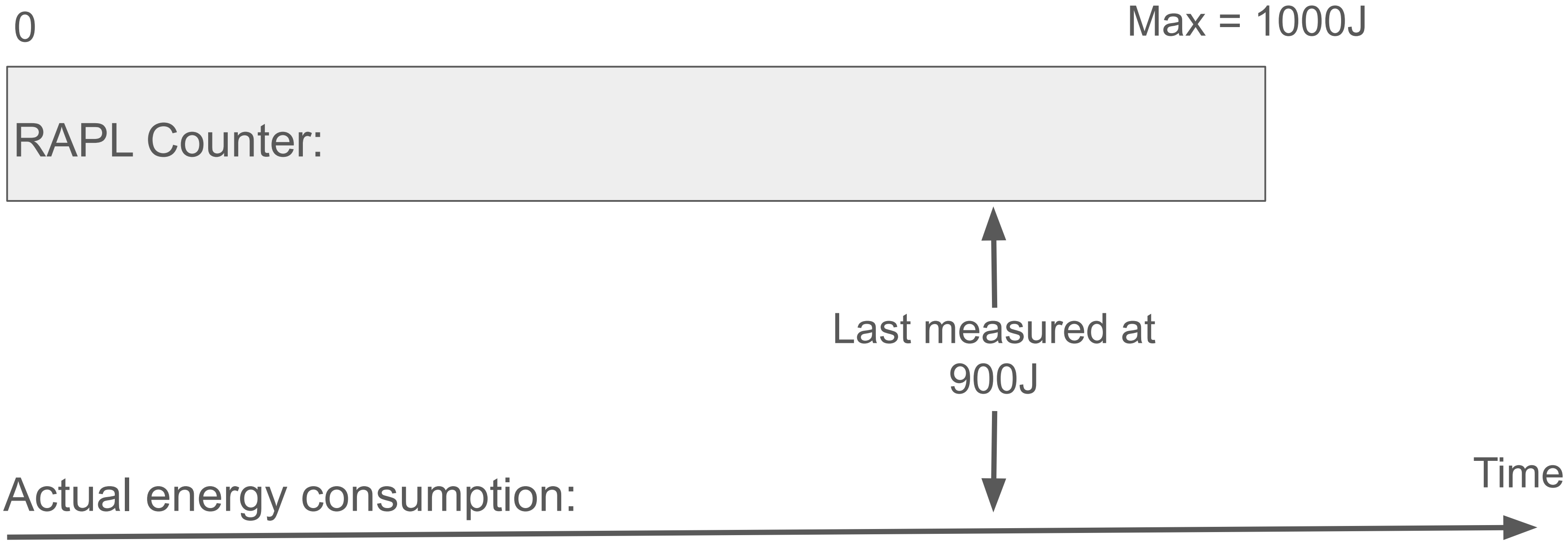
Zeus daemon (`zeusd`) – CPU and DRAM Monitoring tasks



Monitoring Task

RAPL counters wrap around once it reaches `energy_range_uj_max`, typically 20000J

Monitoring Task



0

Max = 1000J

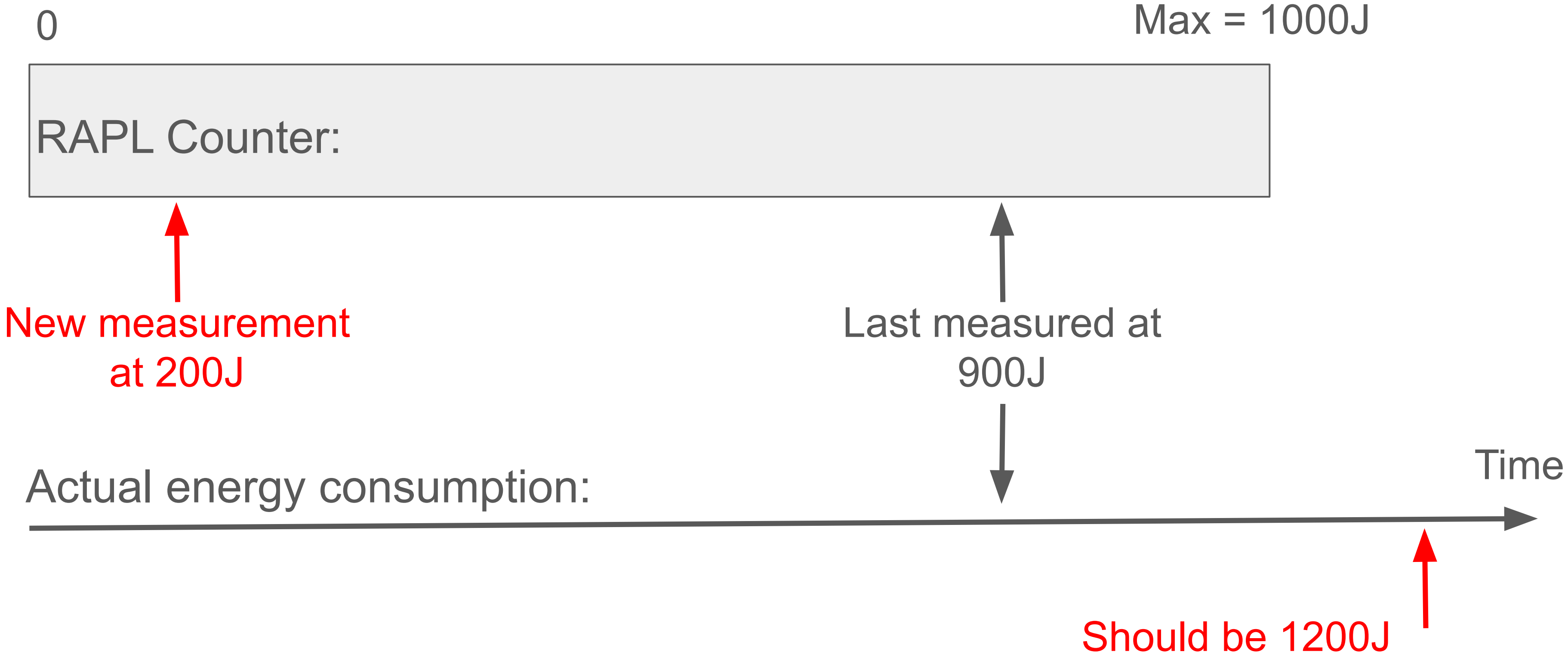
RAPL Counter:

Last measured at 900J

Time

Actual energy consumption:

Monitoring Task



0

Max = 1000J

RAPL Counter:

New measurement at 200J

Last measured at 900J

Actual energy consumption:

Time

Should be 1200J

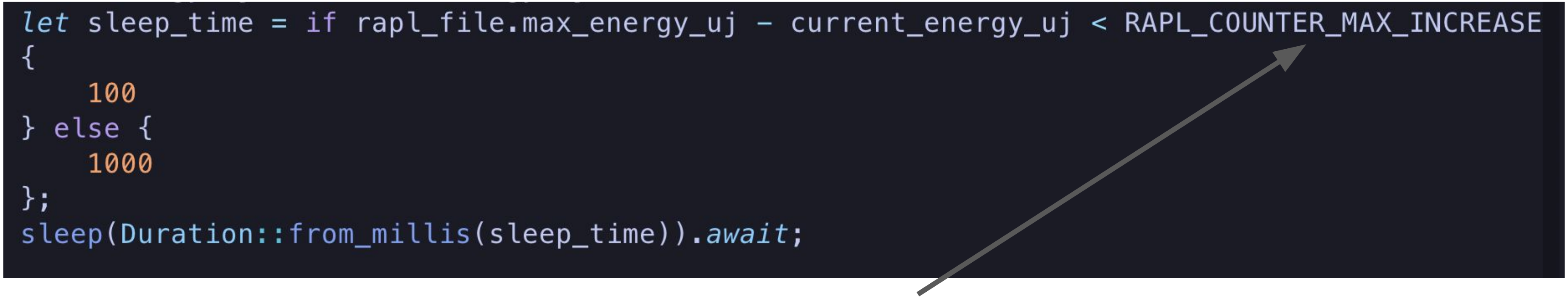
Monitoring Task

Have a separate thread keep track of number of wraparounds

Thread polls RAPL counter every second, updates wrap around count if the new measurement is less than old measurement

Monitoring Task

Polling frequency increases as RAPL counter approaches `energy_range_uj_max`

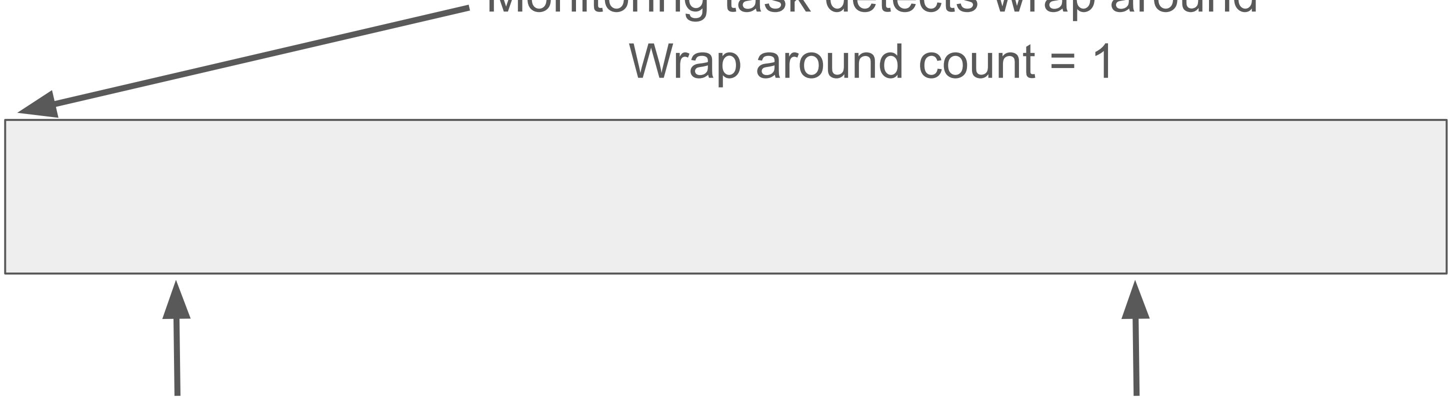


```
let sleep_time = if rapl_file.max_energy_uj - current_energy_uj < RAPL_COUNTER_MAX_INCREASE
{
    100
} else {
    1000
};
sleep(Duration::from_millis(sleep_time)).await;
```

1000 * 0.1 * 1e6

Assuming a maximum power draw of 1000W when polling every 0.1 seconds

Monitoring Task



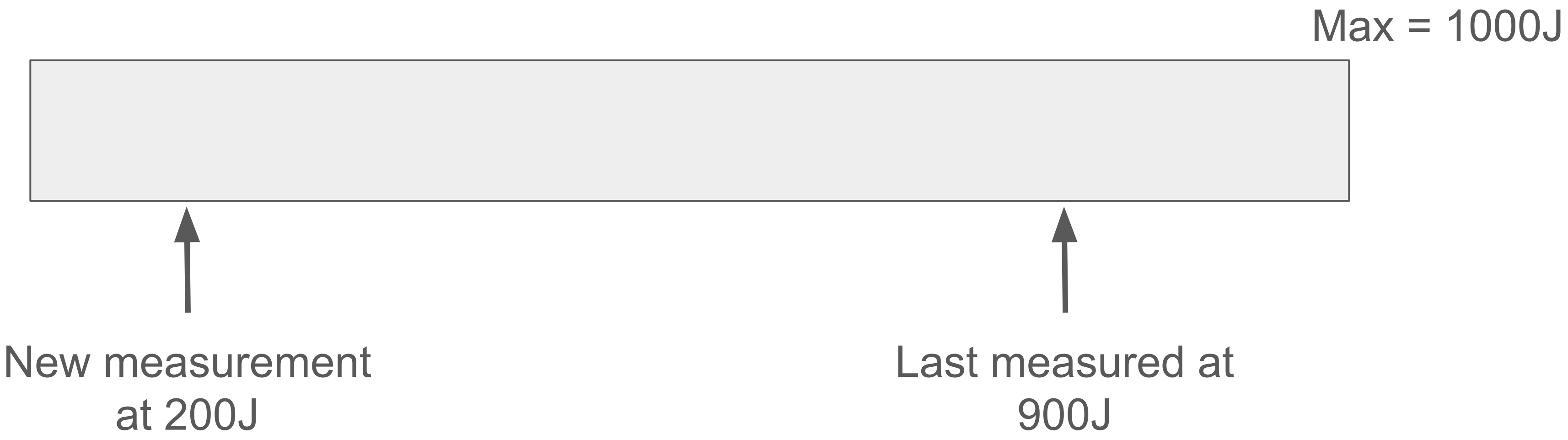
Monitoring task detects wrap around

Wrap around count = 1

New measurement at 200J

Last measured at 900J

Monitoring Task



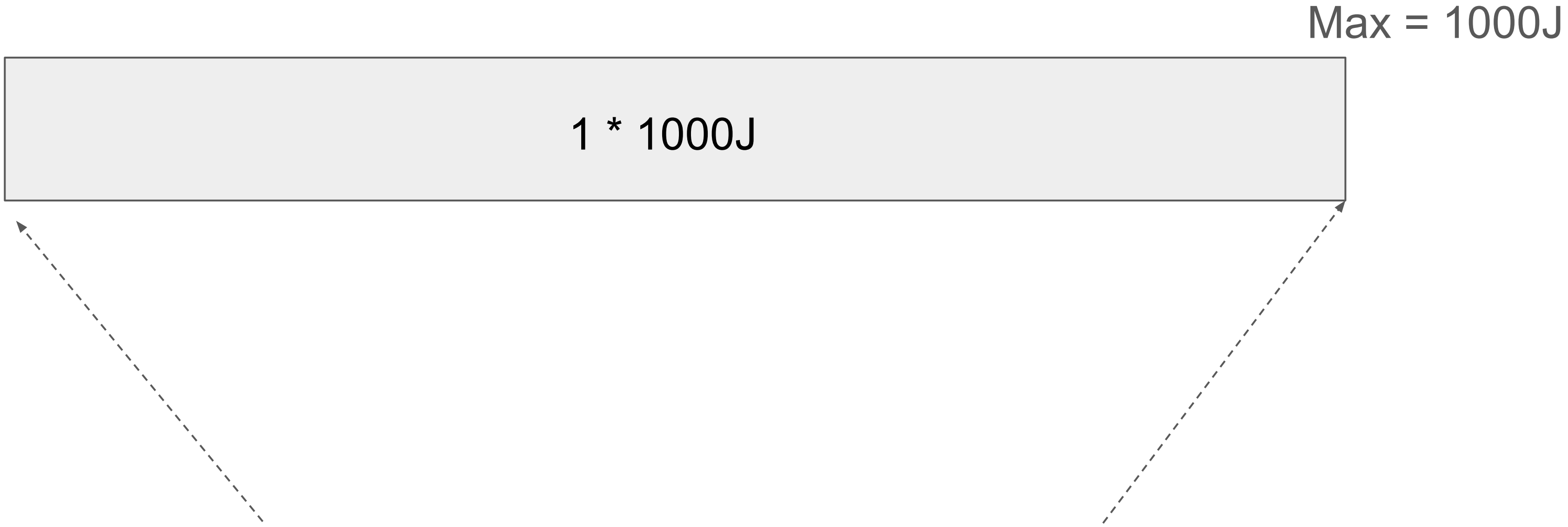
Max = 1000J

New measurement at 200J

Last measured at 900J

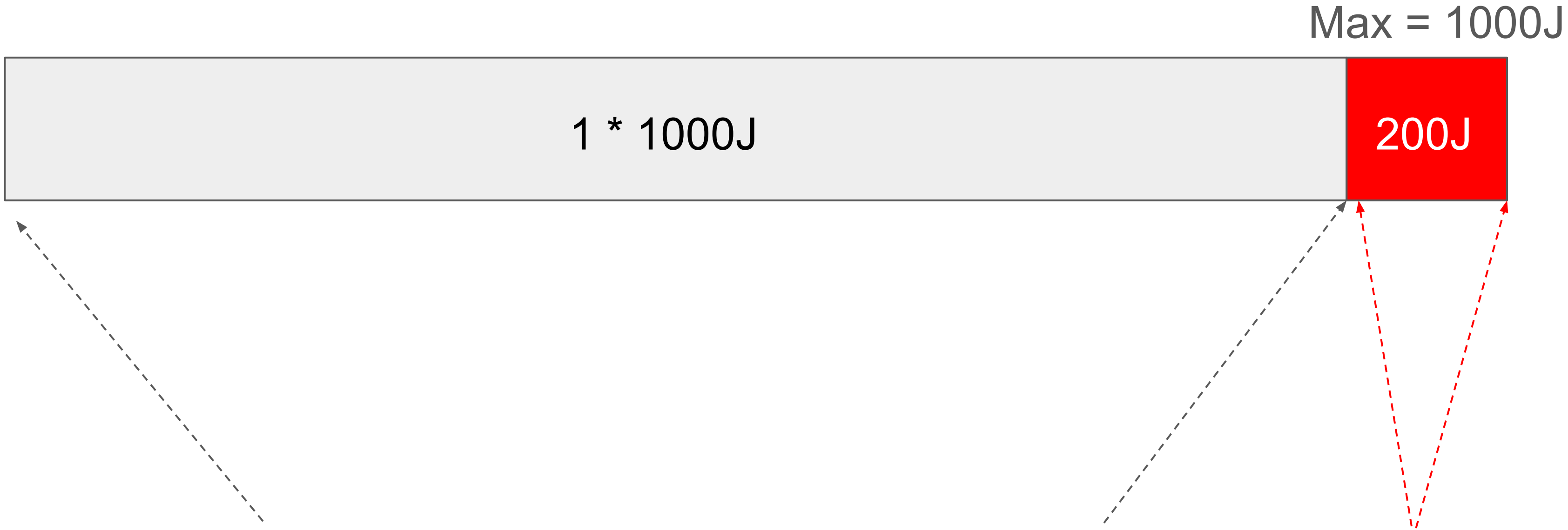
Number of wraparounds * Maximum RAPL counter value + New measurement

Monitoring Task



Number of wraparounds * Maximum RAPL counter value + New measurement

Monitoring Task



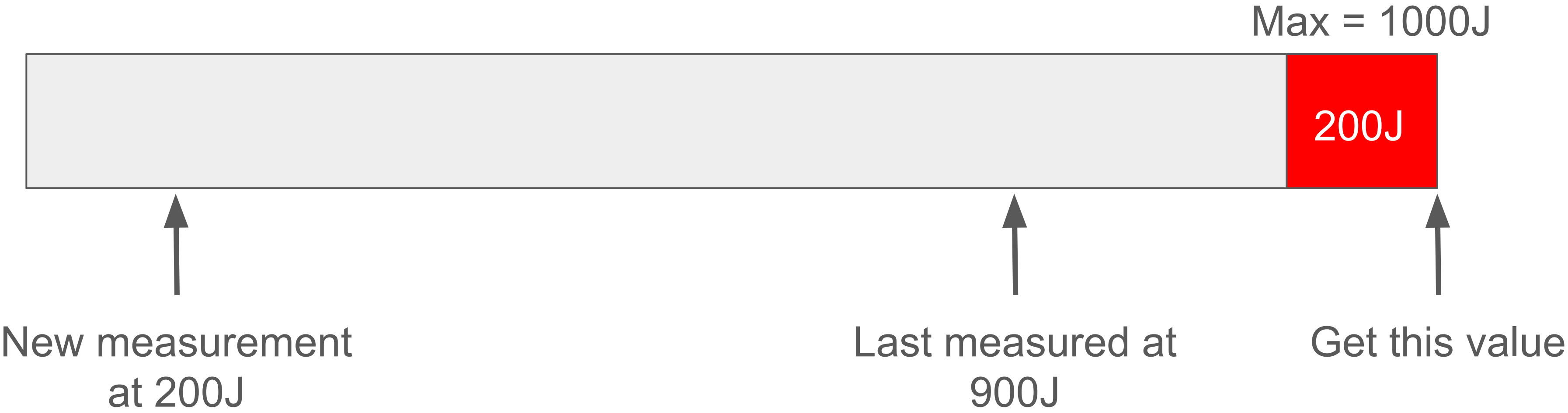
Max = 1000J

1 * 1000J

200J

Number of wraparounds * Maximum RAPL counter value + New measurement

Monitoring Task



Max = 1000J

200J

New measurement at 200J

Last measured at 900J

Get this value

Energy measurement =

New measurement + Number of wraparounds * Maximum RAPL counter value

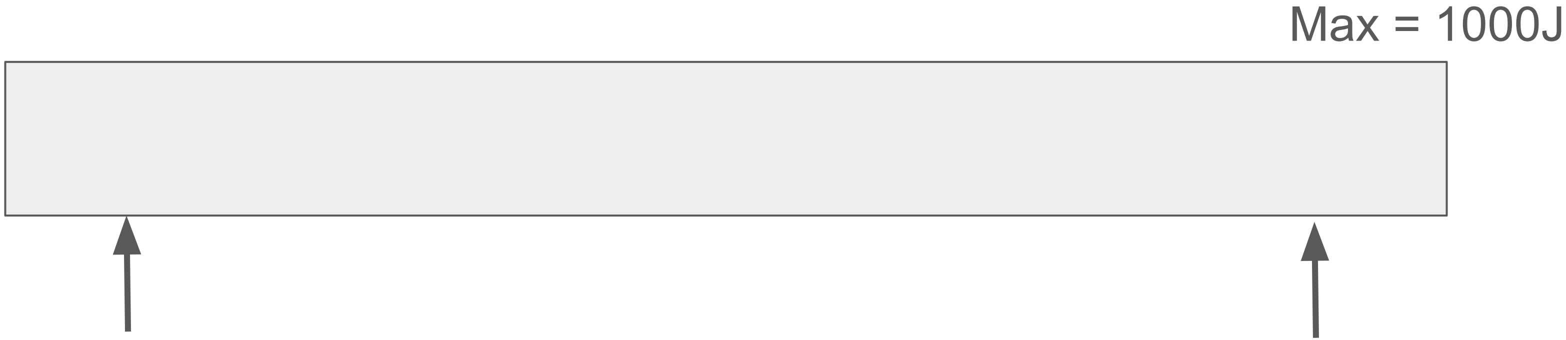
Edge Case

RAPL counter wraps around during a measurement query

1. Read from RAPL counter
2. Get the number of wraparounds
3. Calculate measurement

Edge Case

RAPL counter wraps around during a measurement query



Max = 1000J

Counter at 100J when we get wraparounds

New measurement at 900J

Expected: 100 + 1 * 1000 = 1100J

Actual: 900 + 1 * 1000 = 1900J

Edge Case

Assume counter won’t wrap around twice during a query,

CPU won’t consume > Max energy value during a query

1. Get number of wrap arounds
2. Read from RAPL counter
3. Get number of wrap arounds second time
4. Read from RAPL counter again if there has been a wrap around

Future work

- Use Zeusd in ZeusMonitor
- Find better sleep time in monitoring task.
  - Rather than binary 1 second or 0.1 seconds use running average, gradual decrease, i.e. minimize polling while detecting wrap arounds as early as possible.

Thank you!

Sources

- https://www.devsustainability.com/p/paper-notes-rapl-in-action
- https://www.kernel.org/doc/html/next/power/powercap/powercap.html
- https://web.eece.maine.edu/~vweaver/projects/rapl/
- https://web.eece.maine.edu/~vweaver/projects/rapl/rapl_support.html
- https://hubblo-org.github.io/scaphandre-documentation/explanations/rapl-domains.html