

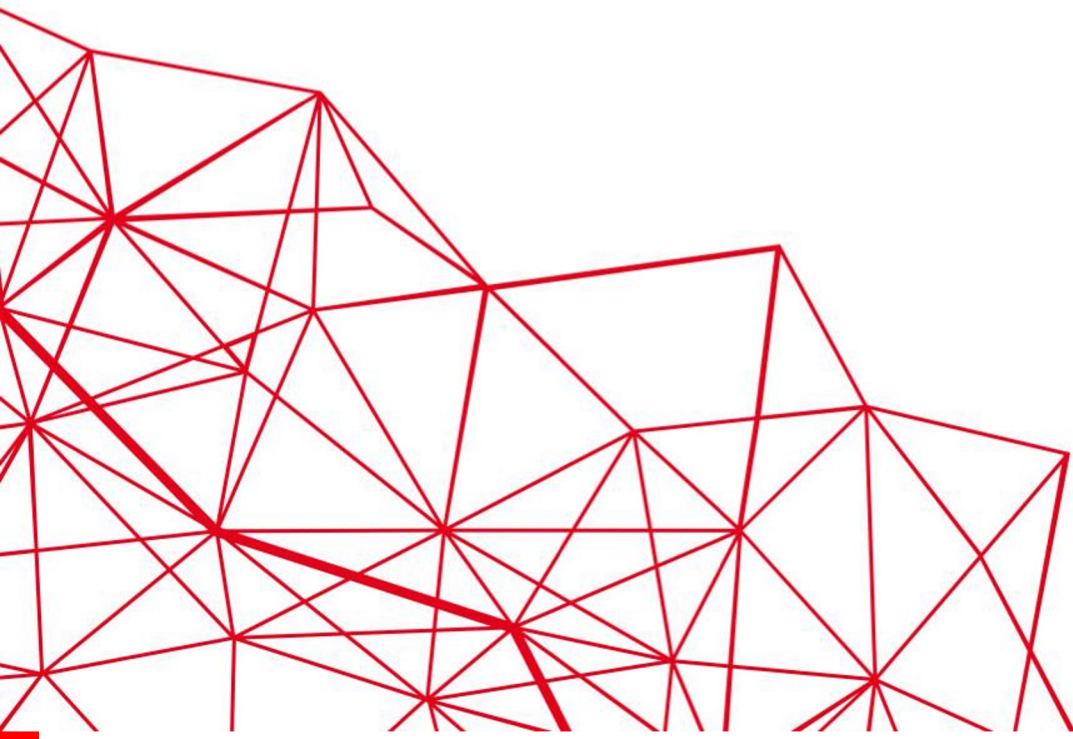
ISC

High Performance

REINVENTING

HPC

MAY 12 – 16, 2024 | HAMBURG, GERMANY



EAR overview

Julita Corbalan (julita.corbalan@eas4dc.com)

Benjamin Czaja (benjamin.czaja@surf.nl)

Martijn Kruijter (martijn.kruijten@surf.nl)

Topics

- EAR overview
- Running jobs with EAR
- Job Monitoring
- Static energy optimization

https://gitlab.bsc.es/ear_team/ear/-/wikis/home

EAR overview

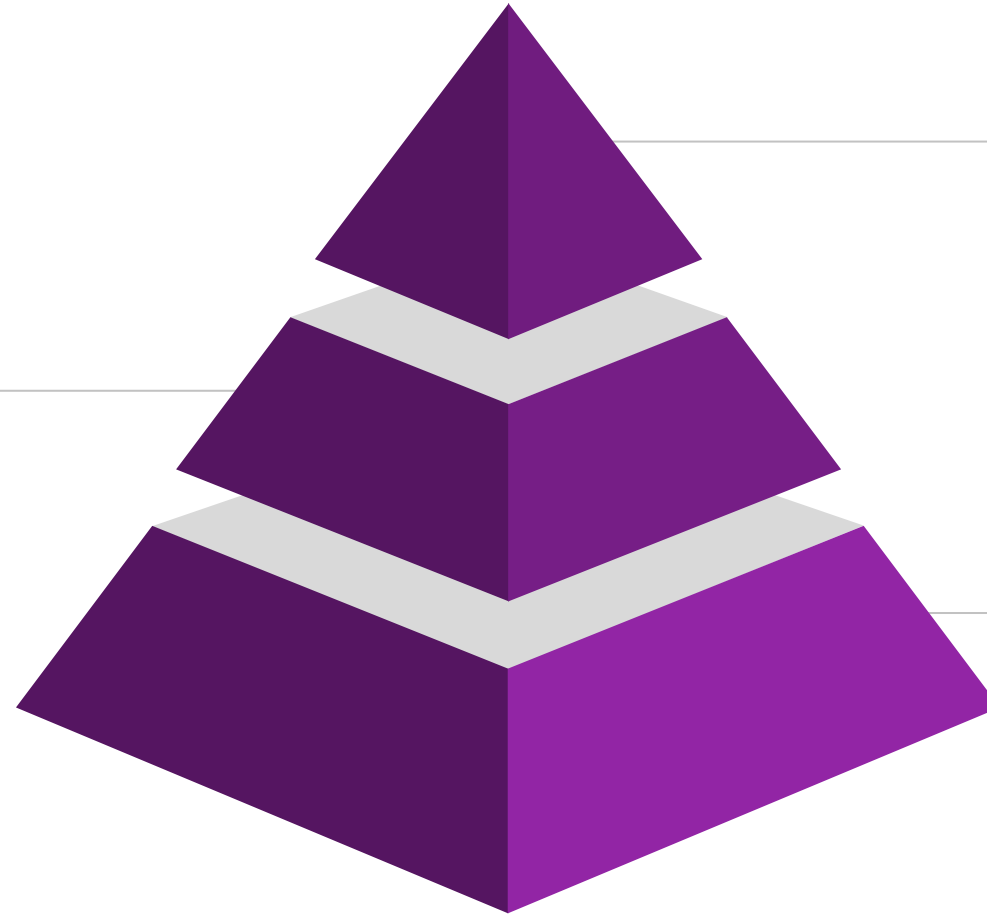
What's EAR: System software for energy management and optimization



Powerful application performance and power monitoring

Runtime library to monitor performance and power dynamically **without any application modification**

2



Energy-efficient system

3

Runtime energy optimization,
Cluster power management and
Cluster and node powercap.

System monitoring and Job Accounting

1

Reports system power
consumption

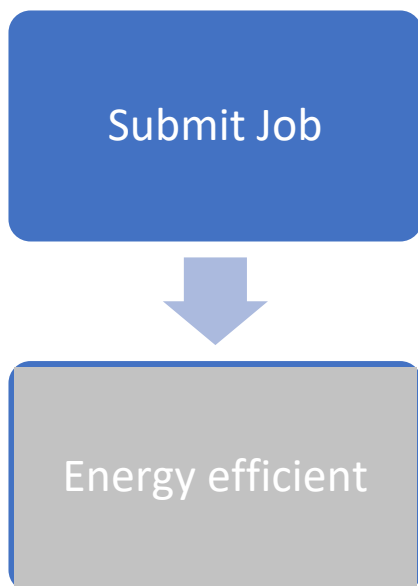
Job energy accounting



What EAR can do for you?

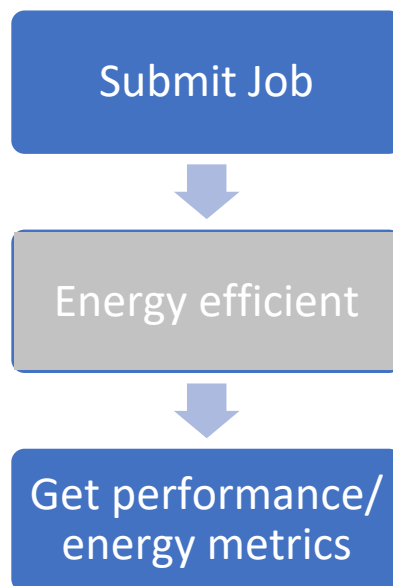


0 effort



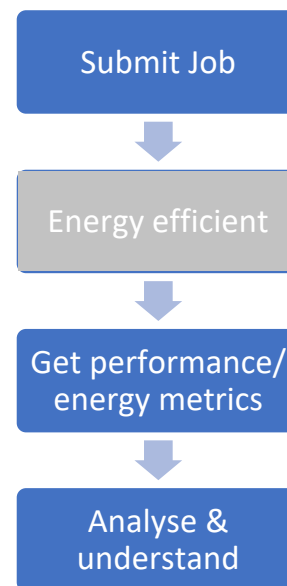
Evaluate
my
app

I want to know



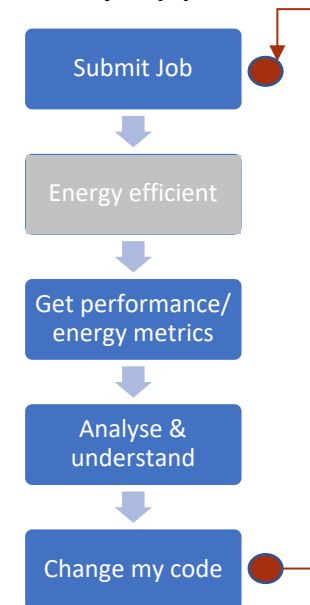
Understand
my
app

I want to
understand



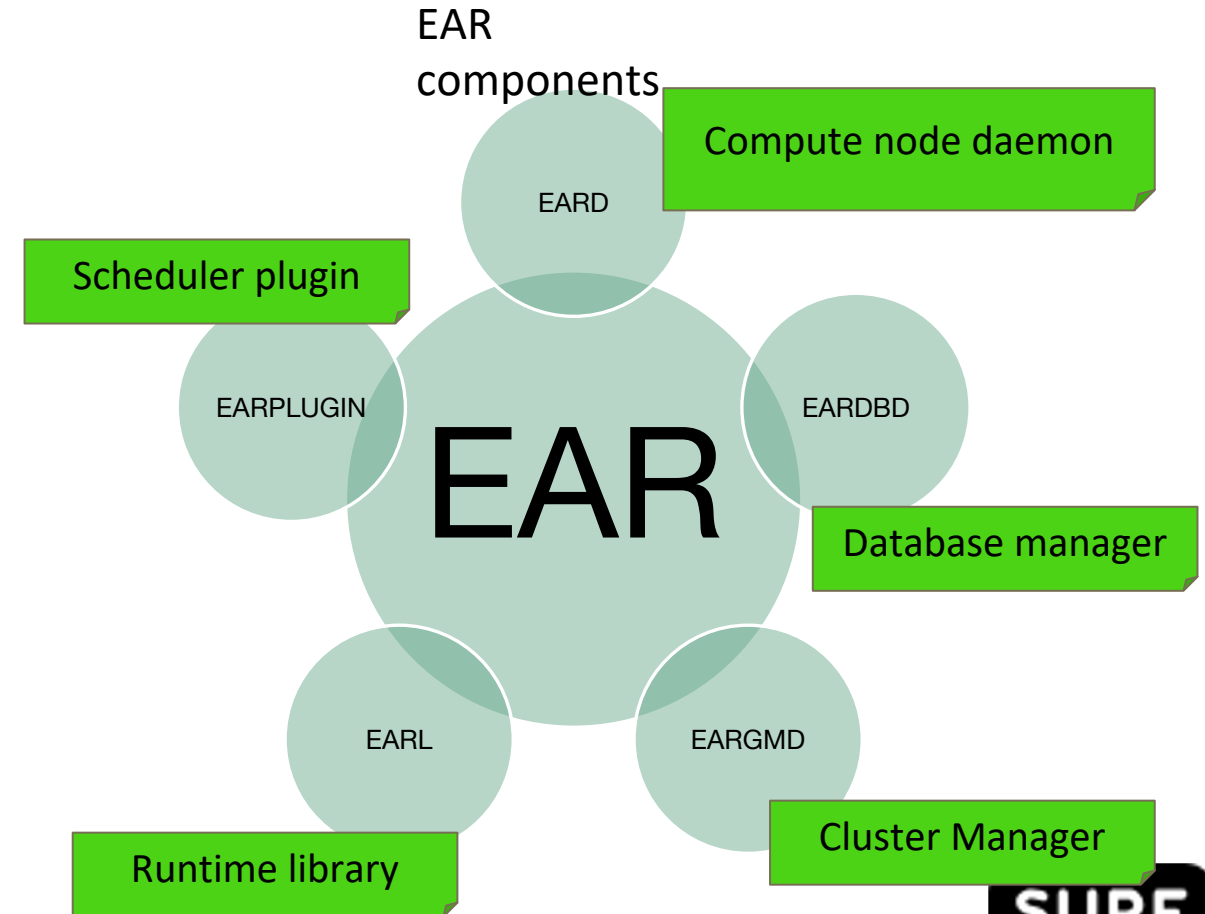
Optimize
my
app

I want to improve
my app



EAR Goals and Components

- To be
 - Easy to use
 - Transparent for users
 - Powerful for developers
- To optimize energy at runtime
- To be flexible and configurable
- Power management

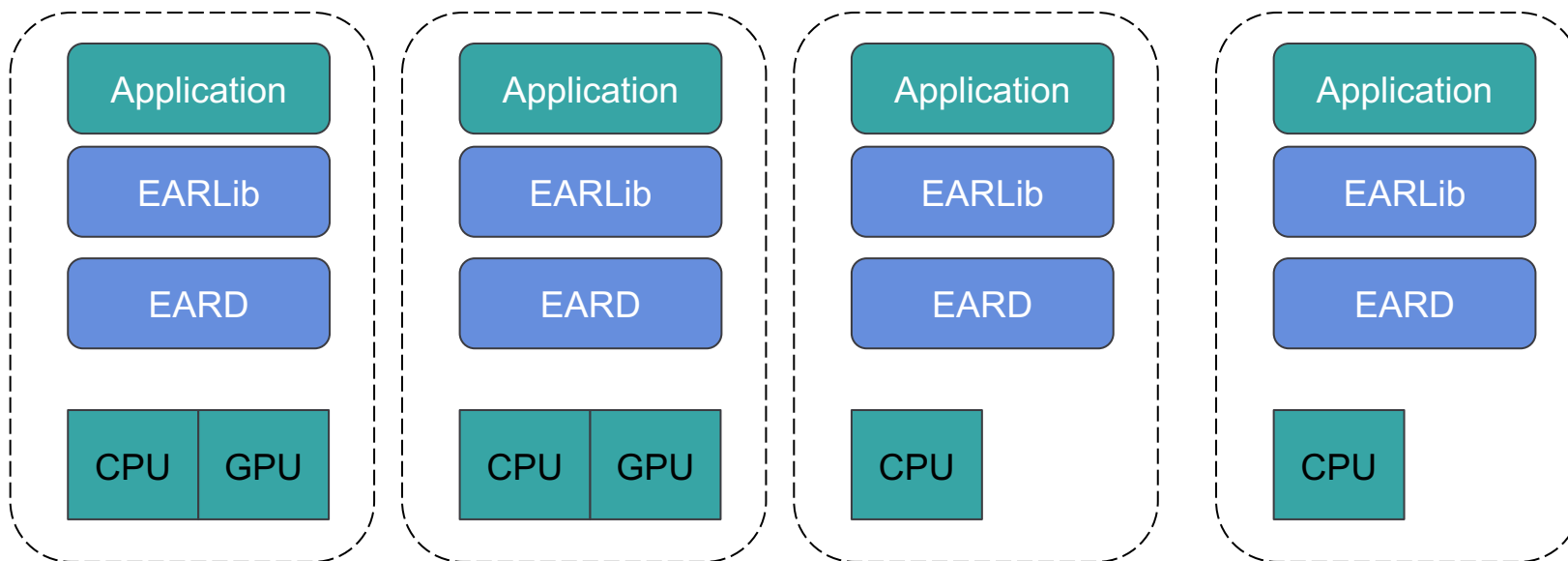
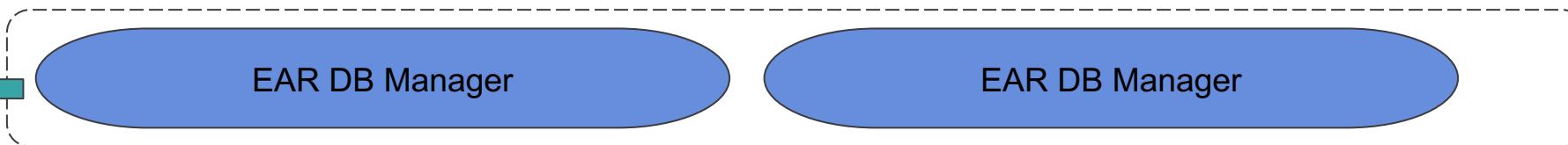
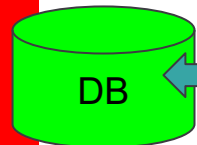
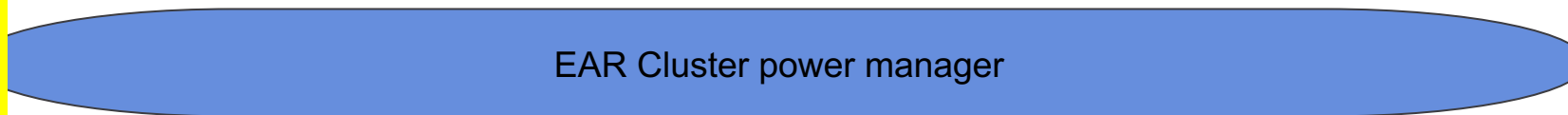


EAR architecture



Cluster

- monitoring
- Optimization
- Power limits



Job Optimization,
monitoring, power
limits



EAS components

EARD: Computational Node Manager

- Linux service running in all the compute nodes (1 x compute node)
- With **root privileges**
- EARD offers
 - **Node monitoring**
 - **Basic Job accounting**
 - **Node powercap**
 - API for local metrics readings and management operations (used by EAR library)
 - API for external power and management
- Applies energy settings described in ear.conf
 - Default policy, frequency
 - Controls EAR privileged users/groups/accounts

EARDBD: Data Base manager

- Linux service running in service nodes
- **Data Base manager.** Connects with DB server
- 1..N EARDBD can be run in the system
- EARDBD aggregates power information
 - Each EARDBD aggregates metrics for all the EARDs contacting with it for a specified period (ear.conf)
 - Each EARDBD reports data to DB in batch messages reducing the number of connections and the cost of queries

EARL: EAR optimization library

- User-level Runtime library
- 100% transparent in most of the use cases (for example MPI applications)
- Application energy and performance monitoring
- Application dynamic energy optimization
- Extensible reporting mechanism
 - Multiple report plugins can be used: DB, CSV, etc

EARGMD: Global/System Power Manager

- System Power manager
- Energy&Power monitoring for the whole cluster
- Cluster powercap

Scheduler plugin

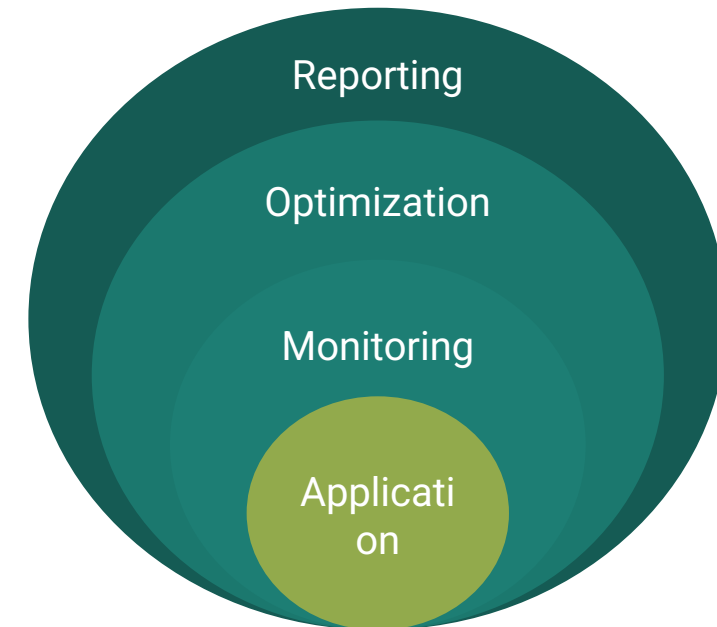
- SLURM SPANK plugin, PBS and OAR supported
- Intercepts job/step creation and
 - connects with EARD to report scheduling events : new job / end job for example
 - Configures the environment for the automatic execution of EAR and environment variables

EAR (runtime) Library

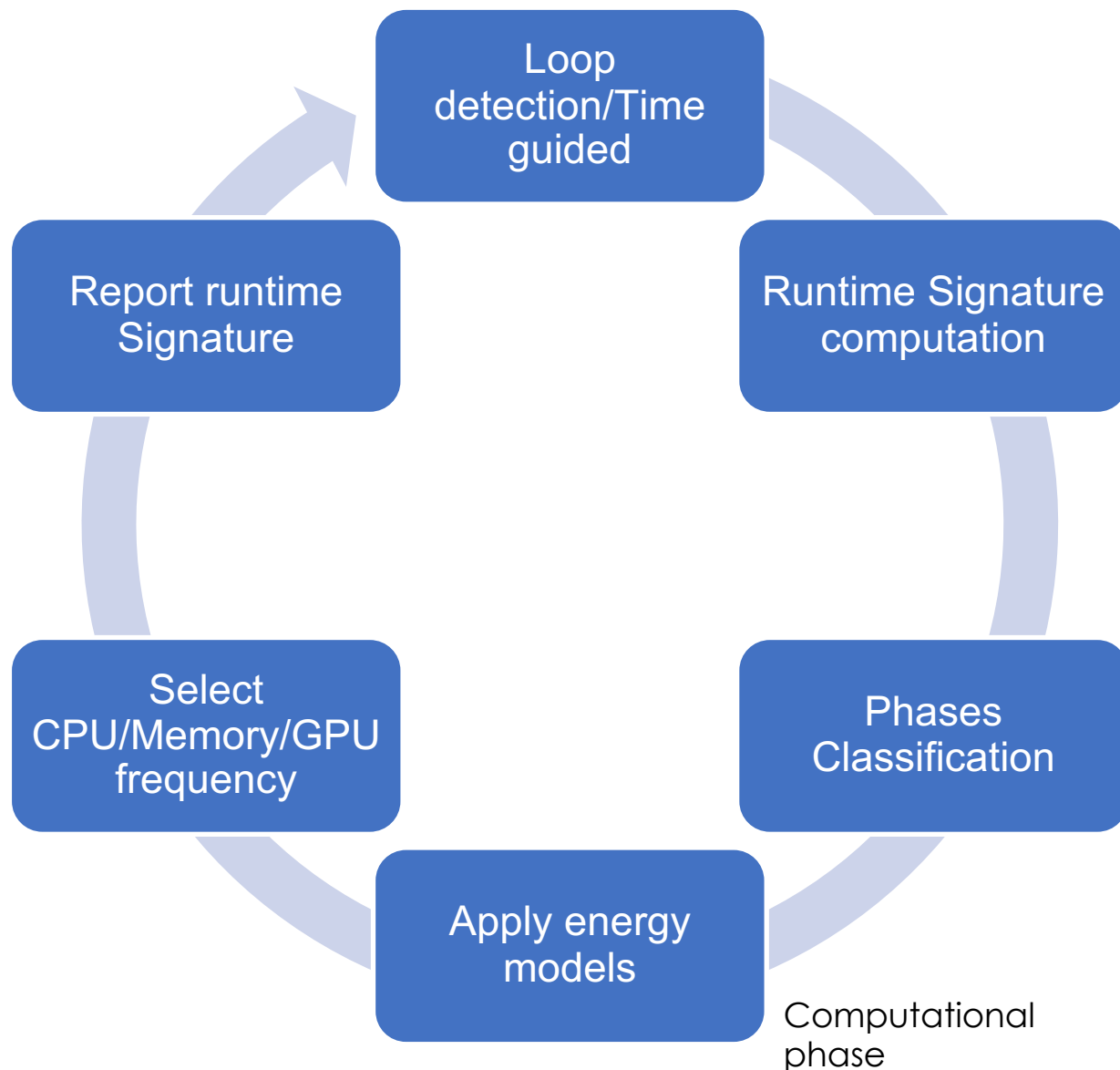
Performance metrics and energy optimization

EAR Library

- User-level Runtime library
- Transparent to users through scheduler plugin (LD_PRELOAD used)
 - Few exceptions need some environment variables: Ex. Singularity
- Application energy and performance monitoring
- Application dynamic energy optimization
- Extensible design based on plugins
 - Multiple report plugins can be used: DB, CSV, etc
 - New policies
 - New energy models
 - ETC



EAS Library lifecycle/stages



- Monitoring
 - Runtime loop detection (MPI only).
 - OR Time guided.
 - Automatic configuration of **chunks** for performance and power accuracy.
- Signatures report
 - Average per jobid/stepid/node.
 - Runtime metrics computed for **chunks**.

chunk = set of consecutive iterations with enough time to compute the power (def=10 sec.)



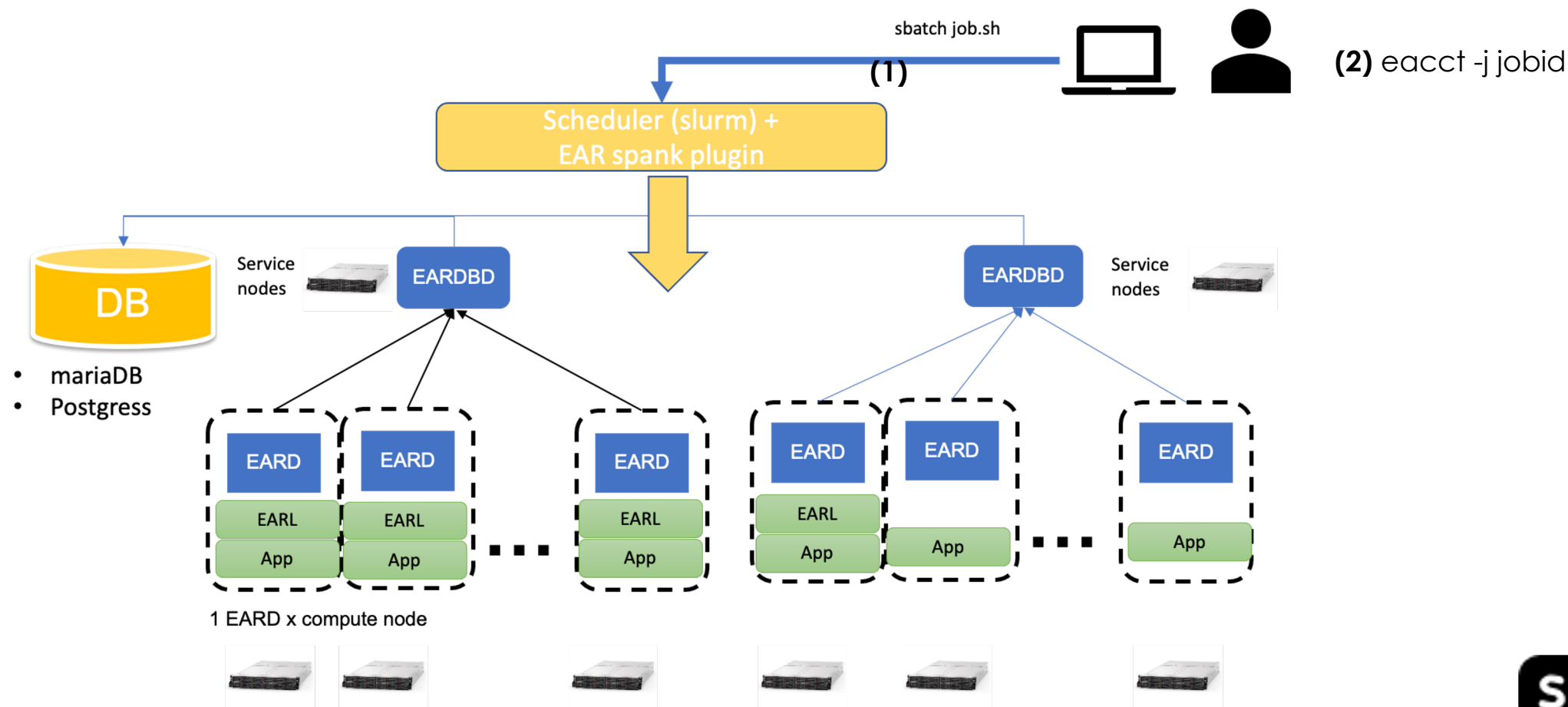
IO phase
GPU bound phase
GPU idle
CPU busy waiting
CPU and GPU

Specific Frequency settings



Running jobs with EAS

Jobs submission/accounting schema



Job submission use cases

- **Without EAR library 100% transparent**
- **With EAR library:** To be 100% automatic EAR library needs
 - Symbol detection
 - Scheduler support (sbatch, srun or salloc)
- erun command or environment variables for not 100% automatic cases

	Use case	Bootstrap	Automatic
MPI	Intel/OpenMPI [+ others]	srun	yes
	Intel [+ others]	mpirun	yes
	OpenMPI [+ others]	mpirun	no : use erun
OpenMP		srun	yes (or use erun)
CUDA		srun	yes (or use erun)
python		srun	yes (or use erun)
Singularity (+any use case)		srun	yes + module or env var support



https://gitlab.bsc.es/ear_team/ear/-/wikis/User-guide#use-cases

Job submission + EAR library options

- Many of the job scripts will work without modification.
- EAR flags in "help"

```
[juliac@int1 ~]$ sbatch --help|grep ear
--ear=on|off          Enables/disables Energy Aware Runtime Library (default OFF)
--ear-policy=type     Selects an energy policy for EAR (monitoring, min_energy, min_time)
--ear-cpufreq=frequency Specifies the CPU frequency to be used by EAR , to be used with monitoring
--ear-user-db=file    Specifies the file to save the job applications metrics (csv format)
--ear-verbose=value   Specifies the level of verbosity (0 default)
```

Memory and GPU frequency also supported with environment variables. See wiki

Special cases (I)

- **MPI+Python is not transparent**, the user (or the module) should define the MPI version because it cannot be detected.
 - `export EAR_LOAD_MPI_VERSION="open mpi"`
 - `export EAR_LOAD_MPI_VERSION="intel"`
- **Singularity**: EAR can be used but EAR paths and env vars must be exported:
`APPTAINER_ENV_XXX`, `APPTAINER_BIND`
 - `export`
`APPTAINER_BIND="$EAR_INSTALL_PATH:$EAR_INSTALL_PATH:ro,$EAR_TMP:$EAR_TMP:rw"`
 - `export APPTAINERENV_EAR_INSTALL_PATH=$EAR_INSTALL_PATH`
 - `export APPTAINERENV_EAR_TMP=$EAR_TMP`
 - `export APPTAINERENV_EAR_ETC=$EAR_TMP`

Special cases (II)

- **OpenMPI**
 - srun recommended
 - use mpirun + erun
- Other use cases/frameworks
 - Force EAR to be loaded with env var
 - export EAR_LOADER_APPLICATION="julia"

Monitoring: EAR metrics

https://gitlab.bsc.es/ear_team/ear/-/wikis/EAR-commands#ear-job-accounting-eacct

Monitoring

- Jobs executed without EAR library (ear = off) report basic job accounting
 - Job/step/node identification
 - Job/step/node execution time
 - Job/step/node energy consumption
- Jobs executed with EAR library (ear =on) report advanced job accounting
 - Job/step/node identification
 - Job/step/node/dynamic performance metrics (measured by EAR library)
 - Job/step/node/dynamic power metrics (measured by EAR library)
- Data is reported in EAR DB

How to get application data

1. **With EAR job accounting command `eacct`:** Command line with pre-defined queries. Multiple filters supported
 - a. average
 - b. per node
 - c. runtime
 - d. pre-selected column in stdout or full data in CSV file
2. **Directly from EAR library**
 - a. csv with timestamp included: `--ear-user-db=filename` (prefix for the file)
 - b. Additional report plugins can be used with env var.
 - i. `EAR_REPORT_ADD=plugin1.so;plug2.so`

Example: https://gitlab.bsc.es/ear_team/ear/-/blob/master/src/report/log.c

Basic MPI example

```
#!/bin/bash

#SBATCH -p rome
#SBATCH -t 00:15:00
#SBATCH --nodes=1
#SBATCH --exclusive

#SBATCH --output=NPB.%j.out
#SBATCH --error=NPB.%j.err
#SBATCH --job-name=NPB

module load 2023
module load foss/2023a

PROJECT_DIR=/projects/0/energy-course

srun --ntasks=128 $PROJECT_DIR/NPB3.4-MZ-MPI/sp-mz.C.x
```

Job submission examples

```
#!/bin/bash
#SBATCH --ntasks=YYY
#### EAR=ON will load all the steps with EAR library
#SBATCH --ear=on

mkdir -p logs
# CASE 1: Default: EAR library on because of headers
srun application
# Runtime metrics reported ON
export EARL_REPORT_LOOPS=1
# CASE 2: mpirun + ear-user-db → CSV file
export I_MPI_HYDRA_BOOTSTRAP_EXEC_EXTRA_ARGS="--ear-user-db=logs/app"
mpirun application
# CASE 3: Using srun + ear-user-db → CSV file
srun --ear-user-db=logs/bt.srun application
# CASE 4: Using erun
module load ear
mpirun -n XXXX erun --ear=on --program="application arg1 arg2...argn"
# CASE 5: EAR library off for this steps
srun --ear=off application
```

eacct: Energy accounting

- SLURM jobid/stepid
- Users can only access its own data
- GPU support is per-cluster, Jobs executed in AMD partition will also show GPU metrics with null values.
- By default, average per job.step metrics: All nodes included. Most metrics are averaged, energy is accumulated.
- Main flags:
 - -l → per node
 - -r → runtime metrics (default is off in snellius. Use `export EARL_REPORT_LOOPS=1`)
 - -c filename → save in CSV format in file

```
[julitac@int3 example]$ eacct -j 1483484
```

JOB-STEP	USER	APPLICATION	POLICY	NODES	AVG/DEF/IMC(GHz)	TIME(s)	POWER(W)	GBS	CPI	ENERGY(J)	GFLOPS/W	IO(MBs)	MPI%	G-POW(T/U)	G-FREQ	G-UTIL(G/MEM)
1483484-sb	julitac	128nodes_16cores	NP	16	2.35/2.60/---	972.00	375.71	---	---	5843040	---	---	---	---	---	---
1483484-1	julitac	128nodes_16cores	MT	16	2.35/2.40/1.47	492.26	373.53	28.44	0.46	2942015	0.0084	250.8	71.7	0.00/---	---	---
1483484-0	julitac	128nodes_16cores	ME	16	2.55/2.60/1.47	460.68	381.00	30.37	0.47	2808271	0.0088	268.2	71.7	0.00/---	---	---

- CPU metrics
 - AVG/DEF/IMC(GHz): **Average** CPU frequency, default frequency and average memory frequency. Includes all the nodes for the step. In KHz.
 - TIME(s): Step execution time, in **seconds**.
 - **POWER: Average DC node power. (in Watts).**
 - **GBS: CPU Main memory bandwidth (GB/second). Hint for CPU/Memory bound classification.**
 - **CPI: CPU Cycles per Instruction. Hint for CPU/Memory bound classification.**
 - ENERGY(J): Accumulated node energy. Includes all the nodes. In Joules.
 - **GFLOPS/WATT : CPU GFlops per Watt. Hint for energy efficiency.**
 - IO(MBs) : IO (read and write) Mega Bytes per second.
 - MPI% : **Percentage of MPI time** over the total execution time. It's the average including all the processes and nodes.
- GPU metrics
 - G-POW (T/U) : Average GPU power. Accumulated per node and average of all the nodes.
 - T = Total (GPU power consumed even if the process is not using them).
 - U = GPUs used by the job.
 - G-FREQ : Average GPU frequency. Per node and average of all the nodes.
 - G-UTIL(G/MEM) : GPU utilization and GPU memory utilization.

Tensorflow: GPU application

```
[julitac@int5 ~]$ eacct -j 5687690
```

JOB-STEP	USER	APPLICATION	POLICY	NODES	AVG/DEF/IMC(GHz)	TIME(s)	POWER(W)	GBS	CPI	ENERGY(J)	GFLOPS/W	IO(MBs)	MPI%	G-POW
(T/U)	G-FREQ	G-UTIL(G/MEM)												
5687690-sb	julitac	run_tensor.sh	NP	1	2.43/2.40/---	2612.00	448.98	---	---	1172725	---	---	---	---
5687690-8	julitac	DenseNet121_disa	MO	1	2.38/2.40/2.19	358.55	897.17	2.03	0.66	321685	0.0000	0.1	0.0	257.92 /257.92 1.410 92%/44%
5687690-7	julitac	DenseNet121_mixe	MO	1	2.38/2.40/2.19	189.30	876.59	0.82	0.67	165936	0.0000	0.2	0.0	238.88 /238.88 1.410 80%/52%
5687690-6	julitac	DenseNet121	MO	1	2.38/2.40/2.19	249.38	890.48	0.59	0.66	222070	0.0000	0.1	0.0	251.56 /251.56 1.410 88%/73%
5687690-5	julitac	VGG19_disable-tf	MO	1	2.38/2.40/2.19	646.86	877.18	2.33	0.52	567419	0.0000	0.1	0.0	238.58 /238.58 1.410 98%/26%
5687690-4	julitac	VGG19_mixed	MO	1	2.38/2.40/2.19	248.40	897.64	0.52	0.67	222972	0.0000	0.1	0.0	257.04 /257.04 1.410 96%/51%
5687690-3	julitac	VGG19	MO	1	2.38/2.40/2.19	261.37	907.44	3.21	0.61	237176	0.0000	0.1	0.0	267.94 /267.94 1.410 96%/59%
5687690-2	julitac	ResNet50_disable	MO	1	2.38/2.40/2.19	302.42	920.38	4.22	0.65	278338	0.0000	0.1	0.0	279.54 /279.54 1.410 94%/43%
5687690-1	julitac	ResNet50_mixed	MO	1	2.38/2.40/2.19	149.26	873.35	0.80	0.66	130356	0.0000	0.2	0.0	233.60 /233.60 1.403 84%/50%
5687690-0	julitac	ResNet50	MO	1	2.38/2.40/2.19	196.33	904.09	0.62	0.65	177504	0.0000	0.2	0.0	261.45 /261.45 1.372 87%/70%

EACCT metrics

- **CPU Bound phases**

- Very low CPI , Cycles per instruction, (less than 0.5)
- Low Memory bandwidth (depends on the architecture)
- Percentage of MPI influences the CPI (MPI waitings are implemented with busy waiting, reduces the CPI)
- High Gflops
- Scale linearly with CPU frequency. Not too much opportunities for energy savings

- **Memory bound phases**

- Medium/High CPI (CPU has to wait for data → increases cycles per instructions)
- High Memory bandwidth
- Medium/High Gflops
- Can take profit of reducing the CPU frequency. Do not scale linearly with CPU frequency

Job Energy Efficiency metric: CPU jobs

- PUE is the ratio of the total amount of power used by a computer data center facility to the power delivered to computing equipment
 - It's not a job metric!!
- Being energy efficient is not consuming less power, is doing an optimal utilization of the power we consume
- For CPU HPC applications, a traditional metric could be the GFLOPS/Watt
 - GFlops characterize the CPU activity
 - Watts the power consumption
- Not valid for
 - Data intensive jobs
 - Non-CPU intensive jobs

Job Energy Efficiency metric: GPU jobs

- GPU metrics reported by default (per-GPU)
 - GPU utilization
 - GPU memory utilization
 - GPU frequency
 - GPU power consumption
- GPU Utilization is not representative enough of the GPU activity
 - NVIDIA GPUs provides more metrics, but many of them are ratios (0...1), for example the FP activity
 - Any action on the GPU increases the GPU
 - New EAS version will include GPU

EACCT metrics: what else can be observed?



- Are nodes homogeneous ? Same signatures in all the nodes (-l)
- Are there phases, is it constant? Are runtime signatures the same? (-r)
- How much time my application is in MPI calls??
- GPU utilization
- Impact on power and performance of changing job configuration
 - Example: Problem with GPU utilization in GROMACS-GPU

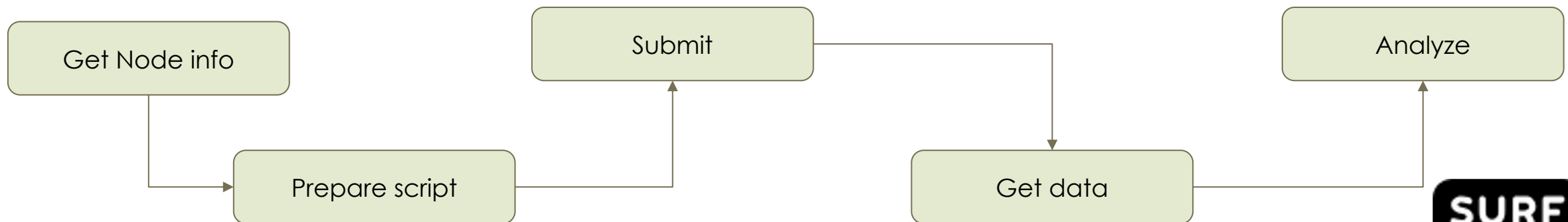
```
[julitac@int3 ~]$ eacct -j 1387089
```

JOB-STEP	USER	APPLICATION	POLICY	NODES	AVG/DEF/IMC(GHz)	TIME(s)	POWER(W)	GBS	CPI	ENERGY(J)	GFLOPS/W	IO(MBs)	MPI%	G-POW
(T/U)	G-FREQ	G-UTIL(G/MEM)												
1387089-sb	julitac	rfm_GROMACS_GPU_NP	1	2.35/2.40/---	6634.00	879.52	---	---	5834703	---	---	---	---	---
1387089-2	julitac	rfm_GROMACS_GPU_MT	1	2.35/2.20/2.02	2886.16	803.91	24.39	0.37	2320201	0.0091	0.1	71.0	333.42/333.42	1.409 10%/0%
1387089-1	julitac	rfm_GROMACS_GPU_ME	1	2.35/2.40/2.10	2880.70	847.76	37.46	0.38	2442131	0.0129	0.1	68.1	359.00/359.00	1.409 14%/0%
1387089-0	julitac	rfm_GROMACS_GPU_MO	1	2.38/2.40/2.19	833.64	1260.21	152.87	0.56	1050561	0.0425	0.4	26.4	650.88/650.88	1.409 73%/0%

Energy optimization

Static energy optimization

- Optimal CPU/Memory/GPU depends on the application, input data, architecture, number of nodes etc etc
- However, you can be interested in applying DVFS in specific case
- With EAS is easy to ask for CPU frequencies
- EAS offers in some architectures more CPU frequencies than available from the OS
- The `enode_info` command reports the EAS technical specification for the computational node



```
$EAR_INSTALL_PATH/bin/tools/enode_info --cpu
EAR CPU info in node tcn2
EAR CPU info Topology: cpu_count      : 128
core_count      : 128
socket_count    : 2
..... // CPU details
EAR CPU info load
EAR CPU info API: EARD
EAR CPU info num devices:128
EAR CPU info list of CPU frequencies
PS0: id0, 2600000 KHz
PS1: id1, 2500000 KHz
PS2: id2, 2400000 KHz
PS3: id3, 2300000 KHz
PS4: id4, 2200000 KHz
PS5: id5, 2100000 KHz
PS6: id6, 2000000 KHz
PS7: id7, 1900000 KHz
PS8: id8, 1800000 KHz
PS9: id9, 1700000 KHz
PS10: id10, 1600000 KHz
PS11: id11, 1500000 KHz
EAR CPU info pstate nominal is 0, CPU freq = 2600000 KHz
EAR CPU info governor CPU[0] = conservative
....
EAR CPU info governor CPU[127] = conservative
EAR CPU info curr CPUF[0] = 2601000
..
EAR CPU info curr CPUF[127] = 2601000
```

```
#!/bin/bash
#SBATCH --job-name=sp
#SBATCH --ntasks=128
#SBATCH --ear=on
module purge
module load 2022
module load iimpi/2022a

export OMP_NUM_THREADS=1
export EARL_REPORT_LOOPS=1
srunch --ear-policy=monitoring --ear-cpufreq=2500000 ./sp-mz.D.128
srunch --ear-policy=monitoring --ear-cpufreq=2400000 ./sp-mz.D.128
srunch --ear-policy=monitoring --ear-cpufreq=2300000 ./sp-mz.D.128
srunch --ear-policy=monitoring --ear-cpufreq=2200000 ./sp-mz.D.128
```

Energy policies: Computational phases

- Monitoring:
 - Application analysis
 - Static energy optimization (Manual CPU/Memory/GPU freq selection)
- Minimize energy to solution (min_energy)
 - EAR **reduces CPU frequency** to save energy with a maximum time penalty
 - Applications start at default frequency and CPU frequency is (potentially) reduced
 - default frequency = nominal frequency
 - Memory frequency selected with a linear search
- Minimize time to solution (min_time)
 - EAR **increases CPU frequency** to minimize time for “*frequency efficient*” codes
 - Applications that scale well with CPU frequency
 - Default frequency = lower than nominal frequency
 - Application will never run at CPU frequency below the default CPU frequency
 - Memory frequency selected with a linear search

Common to both policies

- GPU optimization when GPU idleness.
 - IO phases detected.
 - Turbo can be enabled if configured and CPU bound application.
 - Intra-node Load balance .
-
- GPU frequency selection:
 - Maximum if GPU utilization > 0
 - Minimum if GPU utilization $== 0$ (power consumption is lower.)


```

sbatch --ntasks=192 --partition=genoa sp.D.sh
sbatch --ntasks=192 --partition=genoa -ear-policy=min_energy sp.D.sh

```

KERNEL SP-MZ.D : ROME (Min_energy vs Nominal): Must be the same node for comparison

```
[julitac@int5]$ eacct -j 4896747.0
```

JOB-STEP	USER	APPLICATION	POLICY	NODES	AVG/DEF/IMC(GHz)	TIME(s)	POWER(W)	GBS	CPI	ENERGY(J)	GFLOPS/W	IO(MBs)	MPI%	G-POW (T/U)
4896747-0	julitac	sp	ME	1	2.16/2.60/1.47	336.75	457.62	197.79	0.69	154102	2.9608	0.0	12.0	0.00/---

```
[julitac@int5]$ eacct -j 4896738.0
```

JOB-STEP	USER	APPLICATION	POLICY	NODES	AVG/DEF/IMC(GHz)	TIME(s)	POWER(W)	GBS	CPI	ENERGY(J)	GFLOPS/W	IO(MBs)	MPI%	G-POW (T/U)
4896738-0	julitac	sp	MO	1	2.57/2.60/1.47	329.67	519.01	202.73	0.79	171103	2.6666	0.0	11.0	0.00/---

Thanks!

Julita Corbalan julita.corbalan@as4dc.com, julita.corbalan@bsc.es

Benjamin Czaja (benjamin.czaja@surf.nl)

