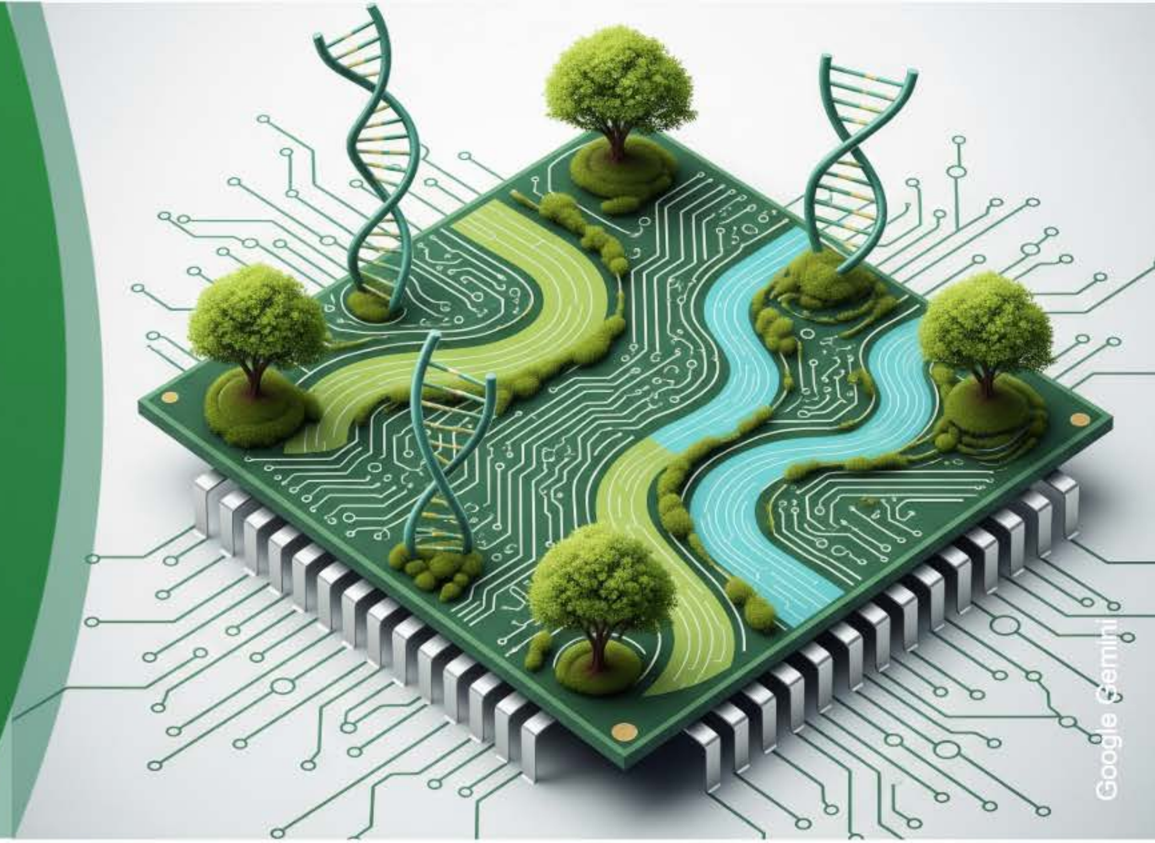
Green and Sustainable Computational Research at EMBL



1. Introduction: What is Green Computational Research?

Environmentally sustainable computational research refers to the practice of designing, developing, and deploying computational methods and systems in a way that minimizes their environmental impact.

This goes beyond using energy-efficient hardware; it encompasses a holistic approach to monitor resource use, reduce carbon footprint and conserve resources, and promote ethical practices throughout the research lifecycle.

Computational research is increasingly central to scientific discovery at EMBL, the energy consumption and resource demands of data centers, high-performance computing (HPC) clusters, and individual workstations make up a significant part of our overall environmental impact.

Addressing these impacts is a crucial part of our Sustainability Strategy and for responsible and sustainable scientific progress more widely.

At EMBL, data centres represent **30%** of the EMBL's electricity use. That's 9 GWh per year (9 million kWh), equivalent to over 3,000 UK households.

Individual projects can have significant carbon footprints: bioinformatics pipelines range from grams to tonnes of CO_2e. A single analysis can exceed the annual target per person of the IPCC* ($2tCO_2e$)!

* IPCC - Intergovernmental Panel on Climate Change

EMBL

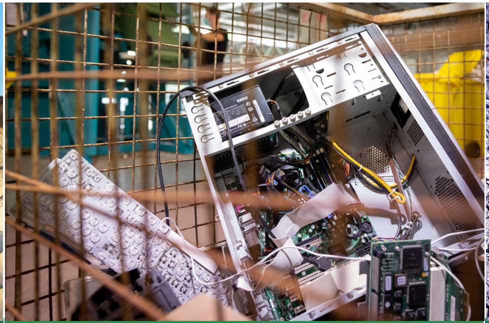
2. The Environmental Impact of Computing

Energy Consumption

Data centers and computational facilities consume vast amounts of electricity, leading to substantial greenhouse gas emissions, especially if powered by fossil fuels.

Resource Depletion

The manufacturing of hardware requires rare earth minerals and other finite resources, often extracted through environmentally damaging processes and often in exploited countries in the Global South.



E-Waste

The rapid obsolescence of hardware contributes to a growing problem of electronic waste, which can leach toxic chemicals into the environment if not properly disposed of.

Water Usage

Cooling systems in large data centers require significant amounts of water.

By adopting green computational practices, researchers can contribute to mitigating climate change, conserving natural resources, and fostering a more sustainable future for science and the planet 🌱

EMBL

3. The Basics - Key Principles of Sustainable Computational Research

- **Energy Efficiency:** Prioritize methods and technologies that minimize electricity consumption during computation, data storage, and data transfer.
- **Carbon Intensity:** Be aware that carbon intensity differs by location, and at different times of the day. Take advantage of grids with high renewable power generation, or times of the day with lower carbon intensities (eg, during the night).
- **Sufficiency:** Challenge the mindset of “the more the better” - instead ask yourself "What is enough to answer my research question?" and design your research around this principle.
- **Algorithm Efficiency:** Select algorithms that achieve desired results with the least computational effort and time.
- **Data Management:** Key outcome is to minimise data movement - data at rest doesn’t consume energy.
- **Resource Optimization:** Make efficient use of hardware, software, and data resources to avoid unnecessary consumption and waste.
- **Hardware Longevity & Reuse:** Extend the lifespan of computing equipment and explore opportunities for reuse or responsible recycling.

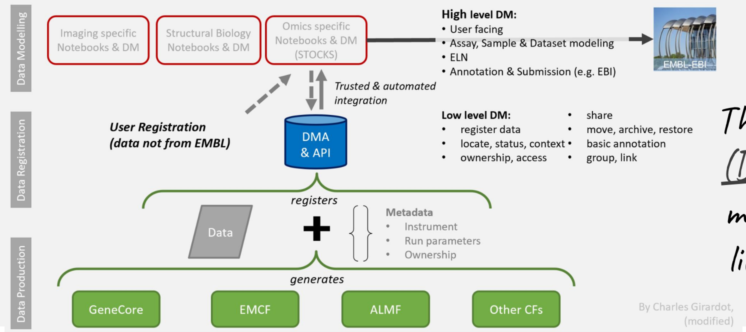
Carbon footprint = energy use X carbon intensity

EMBL

4. The Basics - Data Storage and Management Practices

Data itself has an environmental cost.

- **Data Movement:** Only move data when necessary, and minimise repetitively moving data.
- **Data Reduction:** Store only necessary data. Implement data compression, deduplication, and efficient data formats (e.g., Parquet, HDF5, Zarr) to minimize storage footprint.
- **Data Lifecycle Management:** Regularly review and delete old or unused data. Implement data retention policies.
- **Tiered Storage:** Store frequently accessed "hot" data on faster, more energy-intensive storage (e.g., SSDs) and less frequently accessed "cold" data on slower, more energy-efficient archives (e.g., tape drives, object storage with infrequent access tiers).
- **Local vs. Cloud Storage:** Consider the energy implications of transferring large datasets to and from the cloud versus processing them locally.



The Data Management Application (DMA) is a tool to assist you in managing and documenting your data lifecycle from production to archiving.

5. The basics - Hardware Choices and Management

Your hardware decisions have a direct impact.

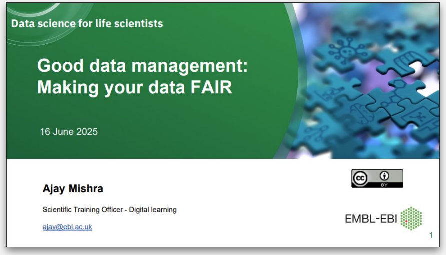
- **Energy-Efficient Hardware:** When purchasing new equipment, prioritize energy-efficient CPUs, GPUs, and storage devices. Look for Energy Star ratings or similar certifications.
- **Green EMBL Tick:** When purchasing from EMBL's standard hardware catalogue, purchase the products which have been given the Green EMBL Tick, which identifies the more sustainable option.
- **Extend Hardware Lifespan:** Avoid unnecessary upgrades. Maintain equipment to ensure it runs efficiently for as long as possible.
- **Shared infrastructures:** when running demanding tasks, use shared computing infrastructures. This avoids purchasing dedicated servers that would be underutilised and operated less efficiently.
- **Virtualization:** Maximize the utilization of physical servers by running virtual machine (VM) servers, reducing the number of physical machines needed.
- **Responsible Disposal:** When hardware is no longer needed, but still work, post it on EMBL's Freecycle database. When hardware reaches end-of-life, ensure it is disposed of in one of EMBL's e-waste bins.

Look for hardware with the **Green EMBL Tick** when shopping from EMBL's hardware catalogue!

6. The Basics - FAIR and Open Science

By adhering to these principles, researchers not only enhance the impact and integrity of their work but also contribute to a more efficient and sustainable global research ecosystem by avoiding duplicated efforts and maximizing the utility of existing computational investments.

- **FAIR principles (Findable, Accessible, Interoperable, and Reusable):** While not directly about energy, adhering to these principles significantly contributes to sustainability by maximizing the value and minimizing redundant efforts in the scientific community.



Data science for life scientists

Good data management:
Making your data FAIR

16 June 2025

Ajay Mishra

Scientific Training Officer - Digital learning

ajay@ebi.ac.uk

EMBL-EBI

1

On-demand training resource: Good data management and FAIR principles.

- **Open Science:** Sharing code, data, and methodologies allows others to build upon existing work, avoiding redundant efforts and promoting efficiency across the research community. For support and to view EMBL's Open Science Policy, contact the Office for Scientific Information Management (OSIM)

7. The basics - Monitoring and Assessing Sustainability in Computational Research

You can’t manage what you don’t measure.

- **Energy Consumption Monitoring:**
  - **Hardware Level:** Use power meters for individual machines or server racks.
  - **Software Level:** Online calculators like Green Algorithms, Python packages like CodeCarbon, or server side tools like GA4HPC can estimate energy consumption of specific processes or code blocks.
- **Carbon Footprint Estimation:** Online tools like CodeCarbon can translate energy consumption into estimated CO_2 emissions, often using regional electricity grid carbon intensity data. The Sustainability Office can also help with this.
- **Resource Utilization Monitoring:** Use system monitoring tools (e.g., htop, nvidia-smi, cloud provider dashboards) to track CPU, GPU, memory, and disk usage.
- **Life Cycle Assessment (LCA):** For hardware, check if your products manufacturer provides a comprehensive Life cycle assessment which evaluates the environmental impacts of hardware from manufacturing to use to disposal.

EMBL

8. Resources

Papers

GREENER principles for environmentally sustainable computational science

Ten simple rules to make your computing more environmentally sustainable

Environmental Impacts of Machine Learning Applications in Protein Science

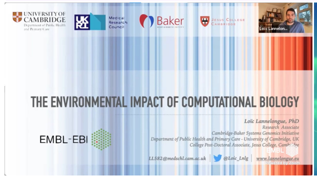
The carbon footprint of bioinformatics

Carbon footprint estimation for computational research

Communities

Green Software Foundation

Lectures


THE ENVIRONMENTAL IMPACT OF COMPUTATIONAL BIOLOGY

EMBL-EBI

Loïc Lannelongue, PhD

US | sussex NEUROSCIENCE

TOWARDS ENVIRONMENTALLY SUSTAINABLE RESEARCH COMPUTING

NICK SOUTER

Tools

Green Algorithms calculator

CODE CARBON

Green Algorithms 4 HPC

EMBL

Green DiSC

EMBL is taking part in Green DiSC, a certification scheme designed to help research groups and institutions address the environmental impact of their computing activities.

The scheme provides a roadmap for sustainable computational research, with different levels of certification (Bronze, Silver, and Gold) that groups can work towards. The criteria are focused on practical, evidence-based actions that can reduce the environmental footprint of research computing.

Is your group taking part?

More information and guidance to enrol is available here.

Optional - Sufficiency - A Deeper Dive

By embracing the principle of sufficiency, you proactively reduce the demand for computational resources, leading to a smaller environmental footprint and fostering a more mindful, "good-enough" approach to your research. Remember to ask yourself ***"When do I have enough to answer my research question?"***

- **Choosing Simpler Models:** Before reaching for a massive deep learning model, consider if a simpler, more efficient statistical model or machine learning algorithm would provide a sufficient level of accuracy and insight. These simpler models require significantly less computational power for training and inference.
- **Working with Representative Data Subsets:** Instead of processing a massive dataset, consider if a smaller, statistically representative sample be used for development, debugging, and initial analysis? This can drastically reduce the computational resources needed for exploratory work.
- **Using Lower Precision:** In many scientific and machine learning applications, using standard 64-bit floating-point numbers is not necessary. Utilizing 32-bit (single-precision) or even 16-bit (half-precision) floating-point formats is good enough and can cut memory usage and computational time, leading to significant energy savings without compromising the validity of the results.
- **Defining Clear Stopping Criteria:** For iterative processes like simulations or model training, establish clear and effective convergence criteria. Stop the computation as soon as a good enough level of accuracy is reached, rather than continuing for marginal improvements.

EMBL

Optional - Code and Algorithm Optimization - A Deeper Dive

The most impactful changes often start with your code.

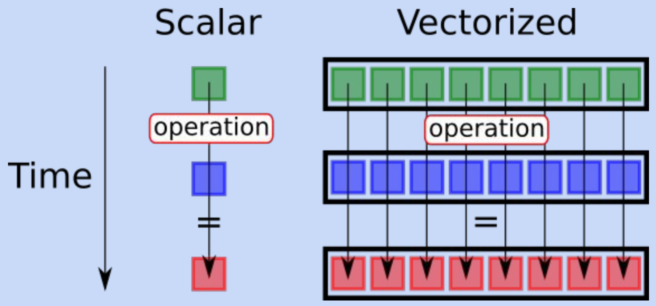
- **Choose Efficient Algorithms:** Before writing code, research and select algorithms known for their computational efficiency (e.g., lower time and space complexity). A less efficient algorithm, even on powerful hardware, can consume significantly more energy.
- **Optimize Code for Performance:**
  - **Vectorization:** Utilize multi-core processors and GPUs effectively which can dramatically speed up computations, reducing the total runtime and thus energy. Libraries like NumPy and TensorFlow are designed specifically to make this kind of parallel computation easy to write and execute.
  - **Memory Management:** Efficiently manage memory to reduce data movement between RAM and CPU cache, or between main memory and storage including cloud based. This can significantly impact energy consumption.
  - **Avoid Redundant Computations:** Cache results of expensive computations if they are reused.
  - **Profile Your Code:** Use profiling tools (e.g., cProfile in Python, perf in Linux) to identify bottlenecks in your code. Focus optimization efforts on these critical sections.
  - **Choose Efficient Programming Languages:** While personal preference and project requirements are key, some languages are inherently more performant (e.g., C++, Fortran, Rust) than others (e.g., Python, R) for raw computation. Consider using optimized libraries or writing performance-critical sections in more efficient languages.
- **Early Stopping/Convergence Criteria:** For iterative algorithms (e.g., machine learning training, simulations), define clear and effective convergence criteria to stop computation as soon as sufficient accuracy is achieved, avoiding unnecessary iterations.

🧐 **Want to know more - check out this wiki - https://wiki.embl.de/cluster/Knowhow**

EMBL

Optional - Vectorization - A Deeper Dive

The process of converting operations that are applied repeatedly to a single operation that is applied to entire arrays or vectors at once.



In a practical sense, it means instead of a computer processor (or GPU) performing a calculation on a single piece of data, it performs the same calculation on an entire block of data simultaneously.

For example, if you have a list of a million numbers and you need to multiply each one by 2, a traditional, non-parallel approach would be to loop through each number one by one. A vectorized or parallel approach would be to tell the processor to multiply all million numbers by 2 in a single instruction, distributing the work across multiple cores or a GPU's thousands of cores.

This drastically reduces the time it takes to complete the task, which in turn reduces the energy consumed.

Graphic - https://lappweb.in2p3.fr/~paubert/ASTERICS_HPC/6-6-1-985.html

EMBL

Optional - Memory Management - A Deeper Dive

Minimizing the energy cost of moving data around within a computer's architecture.

A computer's different types of memory, each with a different level of effort required to access them.

CPU Cache: It's the fastest and easiest to access, but there's very little space.

RAM (Main Memory): It's slower than the CPU cache, but you can fit a lot more there.

Hard Drive/SSD (Storage): It can hold a large amount of stuff, but it takes a long time and a lot of effort to go there and find what you need.

Cloud storage: Can hold a massive amount of data, and accessible from anywhere. But the process is slower due to network latency and consumes a significant amount of energy for both the data transfer itself and the physical infrastructure required to move it.

What your code should do:
By writing code that efficiently manages what data is where, you reduce unnecessary data movement, leading to shorter execution times and lower energy use.
Keep "Hot" Data Close: Your program should be written to keep frequently used data in the fastest memory, like the CPU cache or RAM.
Avoid the "Long Trip": Moving data from storage (SSD/HDD) to RAM is slower and consumes more energy.
Plan cloud storage: minimize how often you need to fetch or save data to cloud storage. For research, this means planning your data access. For example, download a large dataset once, process it locally and then upload only the final, processed results, rather than repeatedly accessing small chunks of data from a cloud.

EMBL

Optional - Profiling Your Code - A Deeper Dive

a process of analyzing the performance of code by measuring the time it takes to perform each function or block.

For environmentally sustainable computing, this is a critical step. By using a profiler, you can pinpoint the specific "hotspots" or bottlenecks in your program. This allows you to focus optimization precisely where they will have the greatest benefit, reducing the total runtime and energy consumption. Without profiling, you might waste time optimizing a part of your code that contributes little benefit to the overall power usage.

Static profiling: analyzing the code without executing it. It helps to identify potential performance issues in the code before it is executed.

Dynamic profiling: analyzing the code while it is running, providing real-time data on the performance which can identify actual performance issues.

Example Profiling Tools
These tools can help you identify performance bottlenecks by measuring execution time and function calls.

For Python:
cProfile, built into Python's standard library.
Scalene: A powerful and modern profiler that measures CPU time, memory, and even estimated energy consumption.

For C++:
Valgrind (with Callgrind): A versatile suite of tools. Callgrind is a profiler that records detailed information on function calls, which can then be visualized with a GUI like KCachegrind.

For Java:
Java VisualVM: A lightweight and user-friendly tool included with the JDK. It provides visual monitoring of CPU, memory, and thread activity.
JProfiler and YourKit: Commercial profilers that offer extensive features for monitoring CPU, memory, and threads with low overhead.

EMBL

Optional - Cloud Computing and HPC Considerations - A Deeper Dive

While cloud providers offer scalable resources, their use requires careful consideration for sustainability.

- **Choose Green Cloud Providers/Regions:** When possible, select data centers located in regions with a lower carbon intensity of power generation. You can compare regions here.
- **Right-Sizing Instances:** Provision only the necessary computational resources (CPU, RAM, GPU). Over-provisioning leads to wasted energy. Utilize auto-scaling features where appropriate.
- **Spot Instances/Preemptible VMs:** For fault-tolerant workloads, using cheaper spot instances can be more resource-efficient as they utilize otherwise idle capacity.
- **Serverless Computing:** For intermittent tasks, serverless functions (e.g., AWS Lambda, Google Cloud Functions) can be more energy-efficient as you only consume energy for the exact execution time.
- **Containerization:** Containers can improve resource utilization by packaging applications and their dependencies, leading to more efficient deployment and less overhead.
- **Schedule Workloads Strategically:** Run non-urgent computations during off-peak hours when electricity demand (and often carbon intensity) is lower.

EMBL

Optional - FAIR Principles - A Deeper Dive

Findable: Ensure your research outputs (data, code, models, methodologies) are easily discoverable by both humans and machines. This involves:

- **Assigning Persistent Identifiers (PIDs):** Use DOIs (Digital Object Identifiers) for datasets and code repositories.
- **Rich Metadata:** Provide comprehensive and accurate metadata that clearly describes the data, methods, and software used, including authors, dates, keywords, and data provenance.
- **Registration in Repositories:** Deposit your outputs in well-indexed, community-recognized repositories (e.g., Zenodo, Figshare, GitHub with proper metadata).

Accessible: Once found, ensure your research outputs can be accessed, ideally through open and standardized communication protocols. This means:

- **Open Access:** Make data and code openly available where possible, respecting ethical and privacy considerations.
- **Clear Access Conditions:** If access is restricted (e.g., for sensitive data), clearly state the conditions and procedures for obtaining access.
- **Standard Protocols:** Use common and open protocols (e.g., HTTP, FTP) for data retrieval:

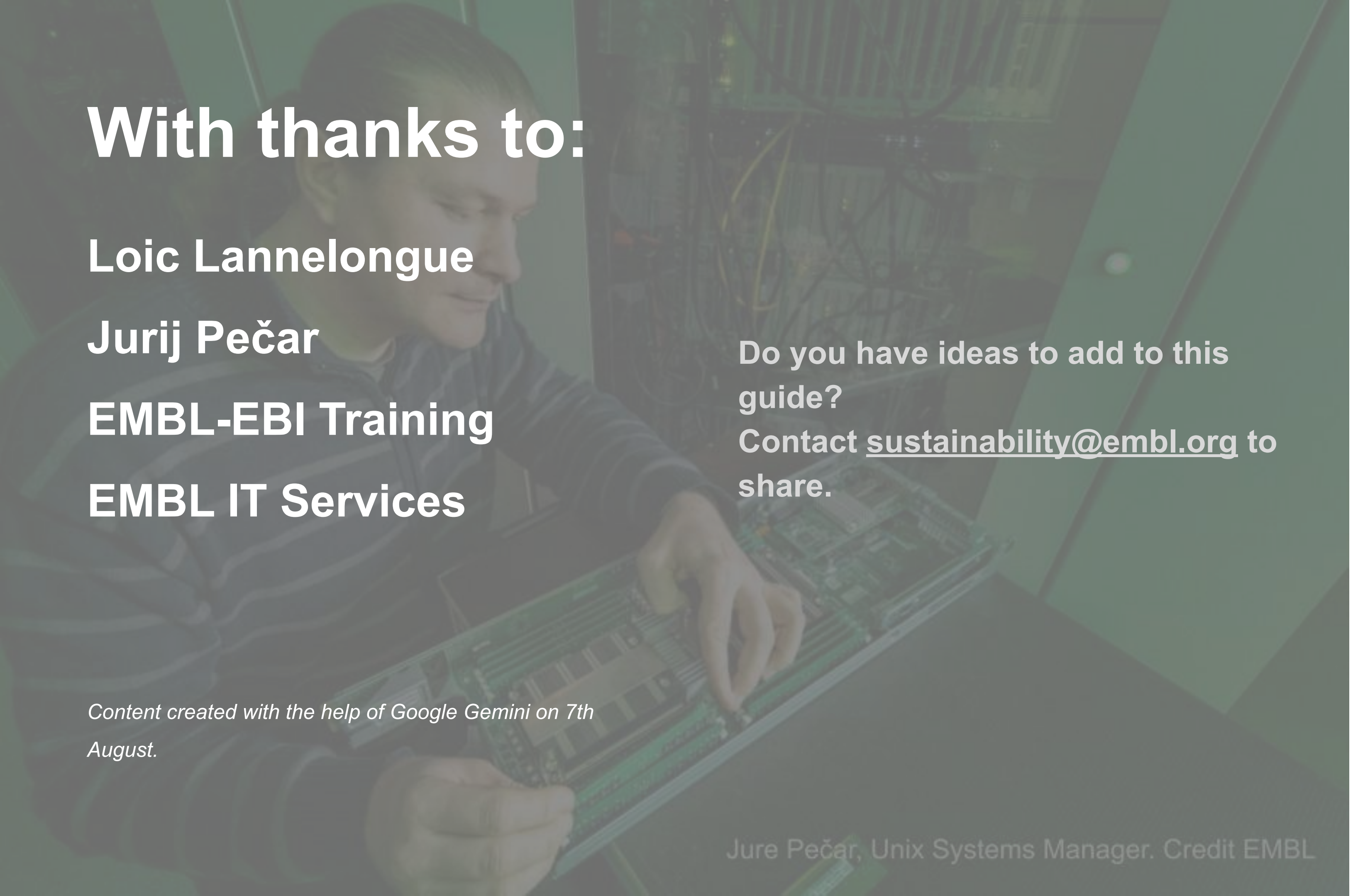
Interoperable: Ensure your research outputs can be integrated with other data and tools, and are compatible with various applications or workflows. This requires:

- **Standard Formats:** Use widely accepted, open, and machine-readable data and code formats (e.g., CSV, NetCDF, HDF5 for data; common programming languages for code).
- **Standard Vocabularies/Ontologies:** Employ shared vocabularies, ontologies, and controlled terms to describe data, enabling consistent interpretation and integration across different datasets.
- **API Documentation:** For software or services, provide clear API documentation to facilitate programmatic access and integration.

Reusable: Promote the reuse of your research outputs by others, including for new research questions or in different contexts. This is achieved by:

- **Clear Licensing:** Apply clear and open licenses (e.g., Creative Commons, MIT, Apache) to both data and code, specifying how they can be used and shared.
- **Detailed Documentation:** Provide thorough documentation for code, data, and methodologies, explaining how they work, their limitations, and examples of usage.
- **Reproducibility:** Ensure that your research can be reproduced by others, reducing the need for redundant experiments and computational efforts. This means providing all necessary components (data, code, environment specifications) to re-run analyses and obtain the same results.

EMBL


Jure Pečar, Unix Systems Manager. Credit EMBL

With thanks to:

Loic Lannelongue

Jurij Pečar

EMBL-EBI Training

EMBL IT Services

Content created with the help of Google Gemini on 7th August.

Do you have ideas to add to this guide?
Contact sustainability@embl.org to share.