Distributed computing services: cluster grids, HPC systems, clouds.

COMPUTE RESEARCH SCHOOL COURSE NTF004F

Evolution of science paradigms

Data exploration (**e-Science**)

- Unifies the rest to explore **Big Data**

Computational science

- Simulations

Theoretical science

- Models

Empirical science

- Descriptive

after Jim Gray

LUND UNIVERSITY

Data tsunami

All scientific research needs **data** (testing models, finding patterns)

All data and information is **digitised**

Modern instruments produce digital data in **huge amounts**

Instruments and data are **accessible by all scientists** on the planet

Modern science is impossible without **global distributed infrastructures** to handle data

Why are Big Data distributed?

There is no Big Storage

- Even Cloud storage is distributed

Preservation

- More copies – better preservation

Instruments are distributed

- Sensors, CT scanners, telescopes, even accelerators

Scientists are many and everywhere

- Local copy is nice to have

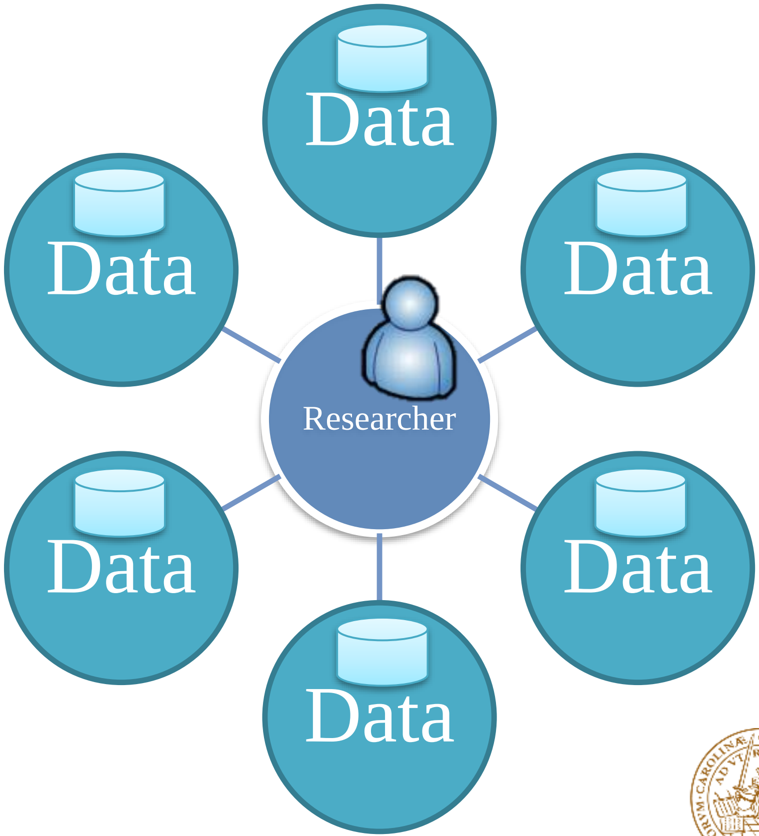
e-Science needs e-Infrastructures

LUND UNIVERSITY

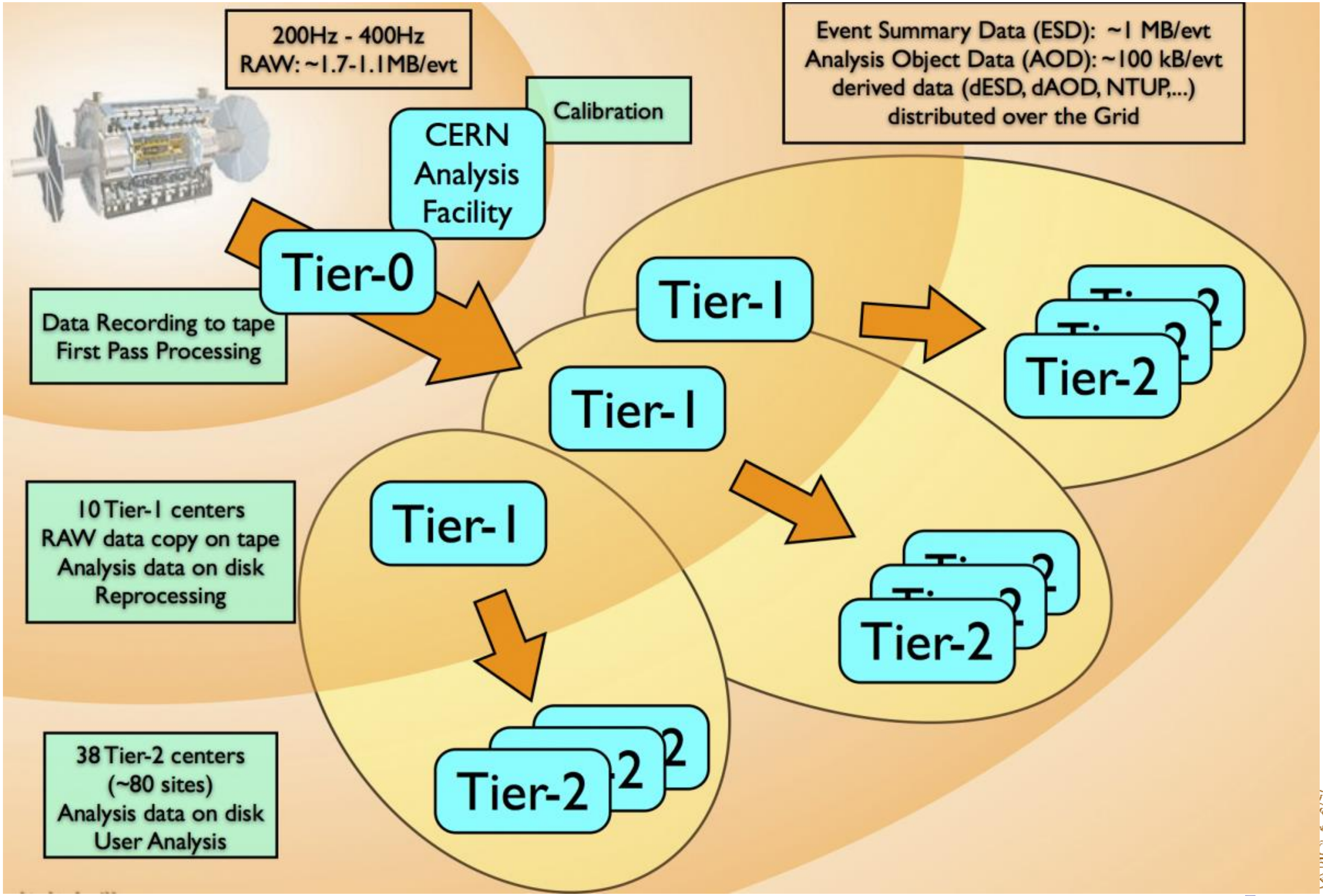
Different data access concepts

Traditional

Distributed data



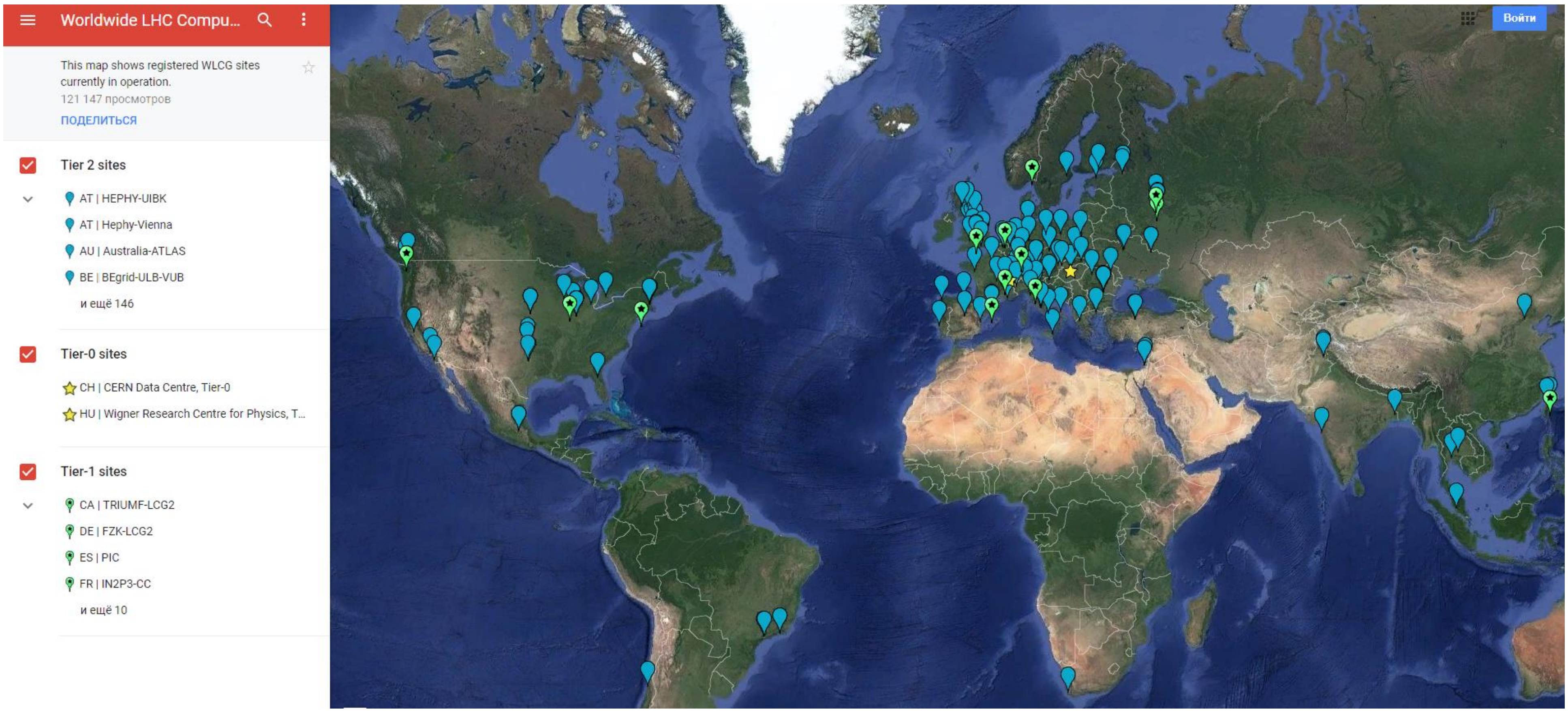
Distributing collected data – CERN way



Graphics by iSGTW

LUND UNIVERSITY

CERN Data and Computing Sites



Check yourself at http://wlcg.web.cern.ch

LUND UNIVERSITY

Recall: scientific computing scenarios

A Infrequent tasks with moderate resource consumption

- E.g. Excel macros, simple image processing etc

B Large batches of similar (simple) independent tasks: serial jobs

- Processing of many images, analysis of accelerator collision events etc

C Lengthy resource-consuming tasks: often parallel jobs

- Pattern recognition, complex system simulation, parameter scanning, lattice QCD
  - Parallel jobs share memory, or exchange information by other means

Typical of massive data processing

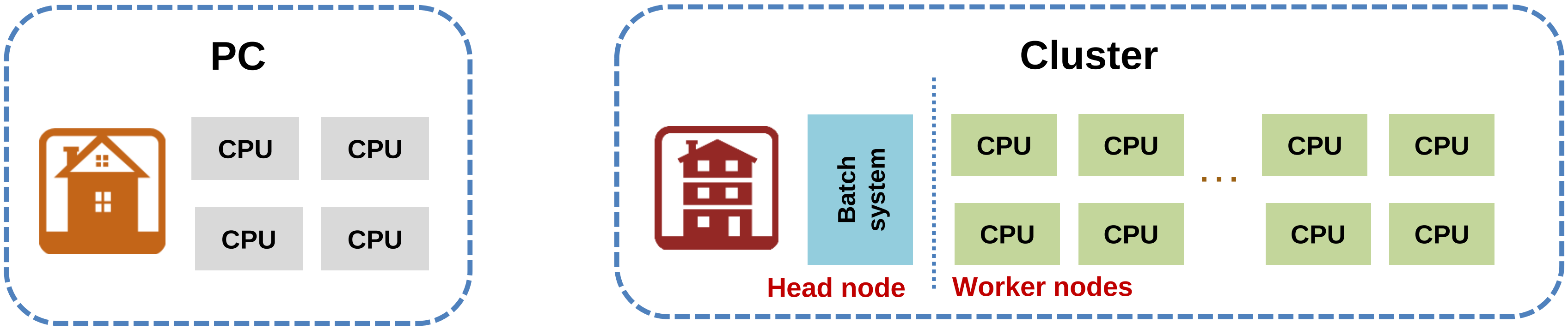
LUND UNIVERSITY

Recall: customized shared service – clusters, supercomputers

- One system serves many users
- One user can use many systems
- Systems are typically provided as public service (by universities and labs)

- Systems are customized, but each can serve many different users
- When many different systems jointly offer common services (login, storage etc), they create a computing *Grid*

From PCs to Clusters and HPC systems



- Clusters started as loosely coupled PC boxes
  - Now industry solutions exist
    - » Even virtual
- HPC: High Performance Computing systems
  - Supercomputers or very advanced clusters
    - » Shared memory, fast interconnect etc

The workhorse of scientific computing: clusters

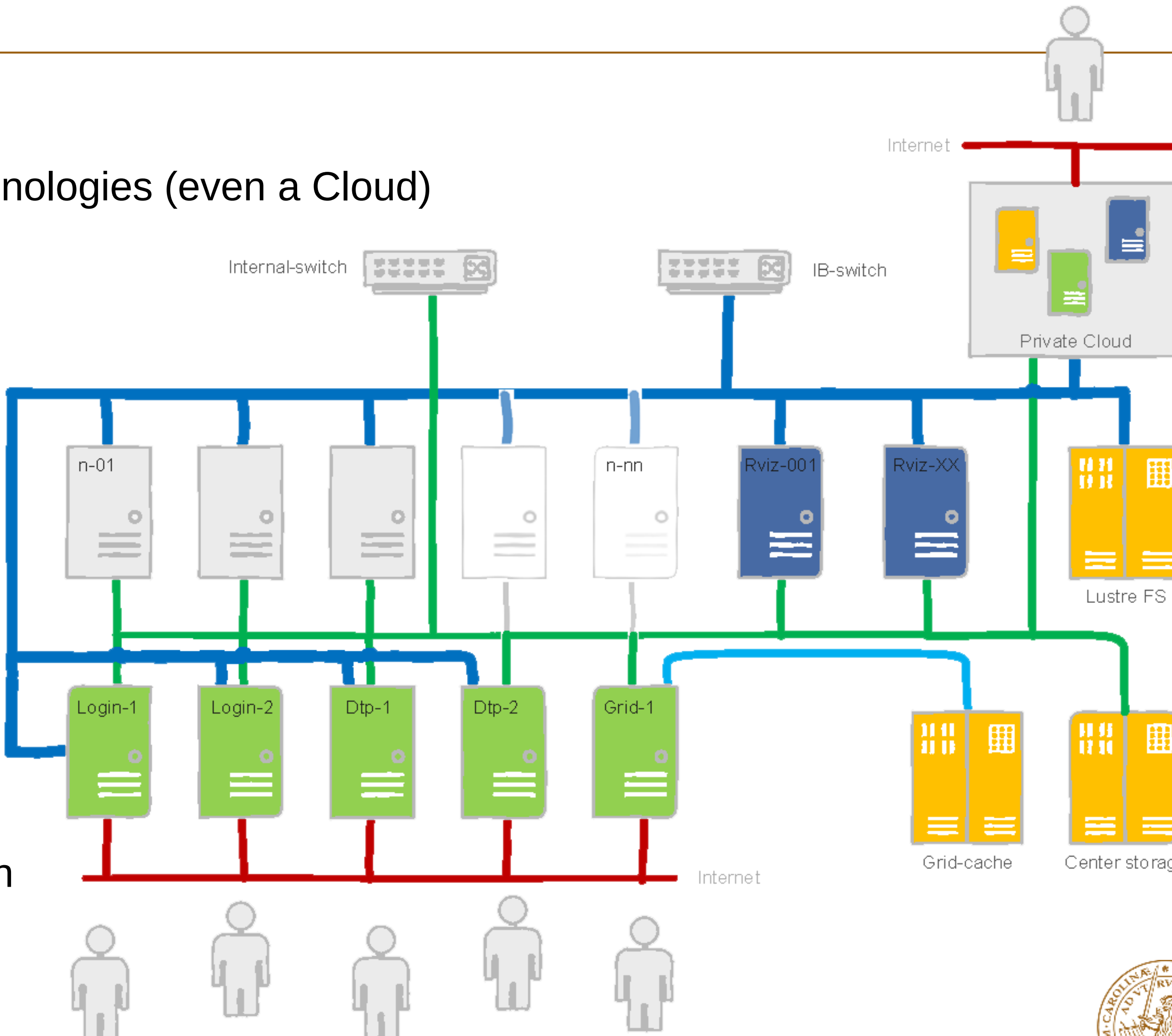
- Computing facilities in universities and research centers are usually Linux <u>clusters</u>
  - Some supercomputer-grade facilities are often clusters, too
- A cluster is a (comparatively) loosely coupled set of computing systems presented to users as a single resource
  - 0-level **distributed computing**, because of loose coupling
  - A typical cluster has a <u>head node</u> (or a few) and many <u>worker nodes</u>
    - » A node is a unit housing processors and memory – a server
  - Distribution of load to worker nodes is orchestrated by means of Local Resource Management Systems (a.k.a. <u>batch systems</u>)
    - » Many batch systems exist on the market: SLURM, PBS, SGE, HTCondor etc
- Every cluster is a heavily customised resource built for a range of specific applications

Clusters in the LUNARC center

Photo: Gunnar Menander, LUM

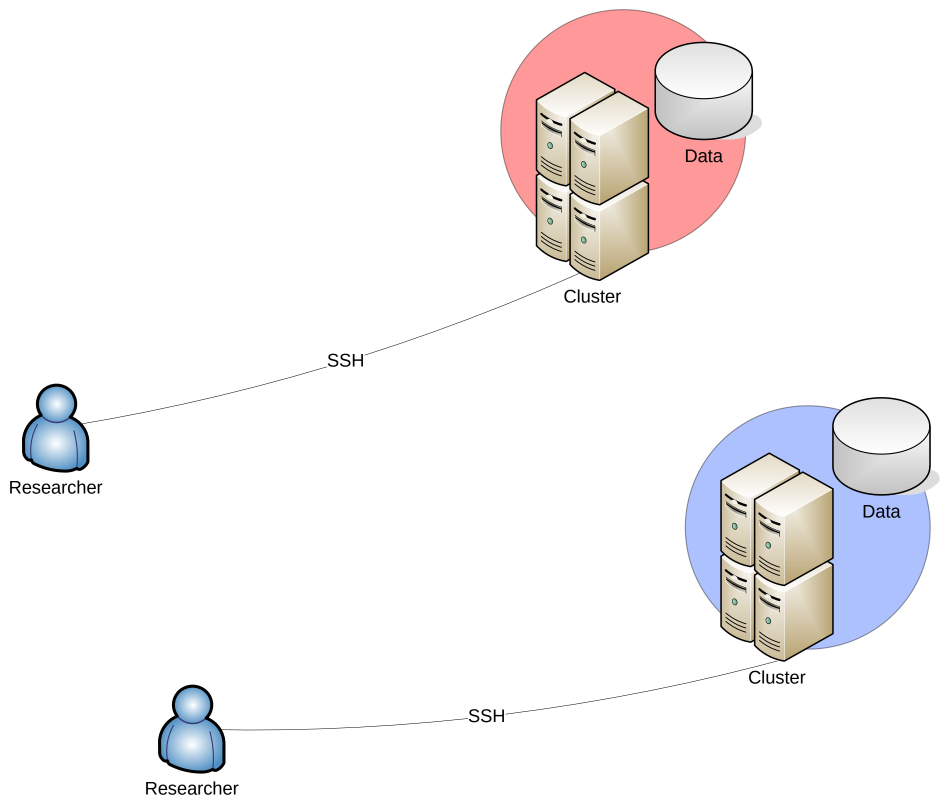
Aurora cluster at LUNARC

- Combines many different technologies (even a Cloud)
- Offers many different services
  - Computing (of course)
  - Storage
  - Remote desktop
  - Etc
- 180 base nodes + over 50 nodes owned by research groups
  - Each node has 2 Haswell processors, 20 cores each



Graphics by J. Lindemann

Typical workflow on clusters



- Users connect to the **<u>head node</u>**
  - Typically, using Secure Shell - SSH
- Necessary software is installed
  - For example, your own code
    - » Usually centrally by admins
- Specialised scripts are used to launch tasks via **batch systems**
  - A task can be anything, from adding 2 and 2, to bitcoin mining
  - A single task is called a **job**
- Data are placed in internal storage – not really distributed
  - Cluster worker nodes often have no network

LUND UNIVERSITY

Jobs and queues

- A batch system is software that schedules computational tasks to worker nodes according to given criteria and requirements
- A single unit of scheduling is called a **job**; some job requirements are:
  - A job can use a single core (serial job), or several cores at once (parallel job)
  - Consumes CPU time and astronomic (wall-clock) time
    - A well-parallelized job will consume less wall time, but a comparable CPU time to a serial one
  - A job also consumes memory and disk space
  - A job may do intensive input/output operations (data processing)
  - A job may require public network connectivity (for example, for database queries)
- When there are more jobs than resources, **queue** management is needed
  - A cluster may have several queues for different kinds of jobs (long, short, multicore etc)
  - A queue is actually a persistent partition of a cluster, exists even if there are no jobs

Different tasks use different jobs

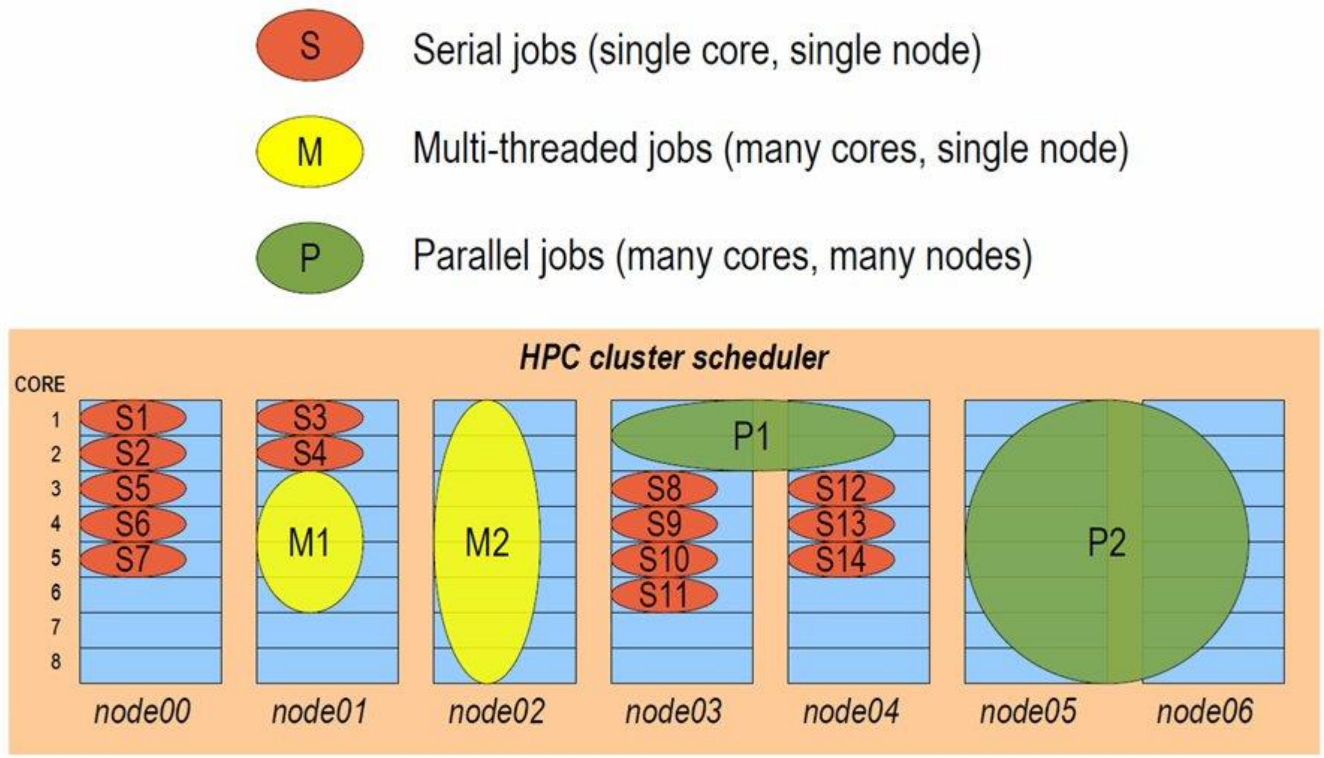
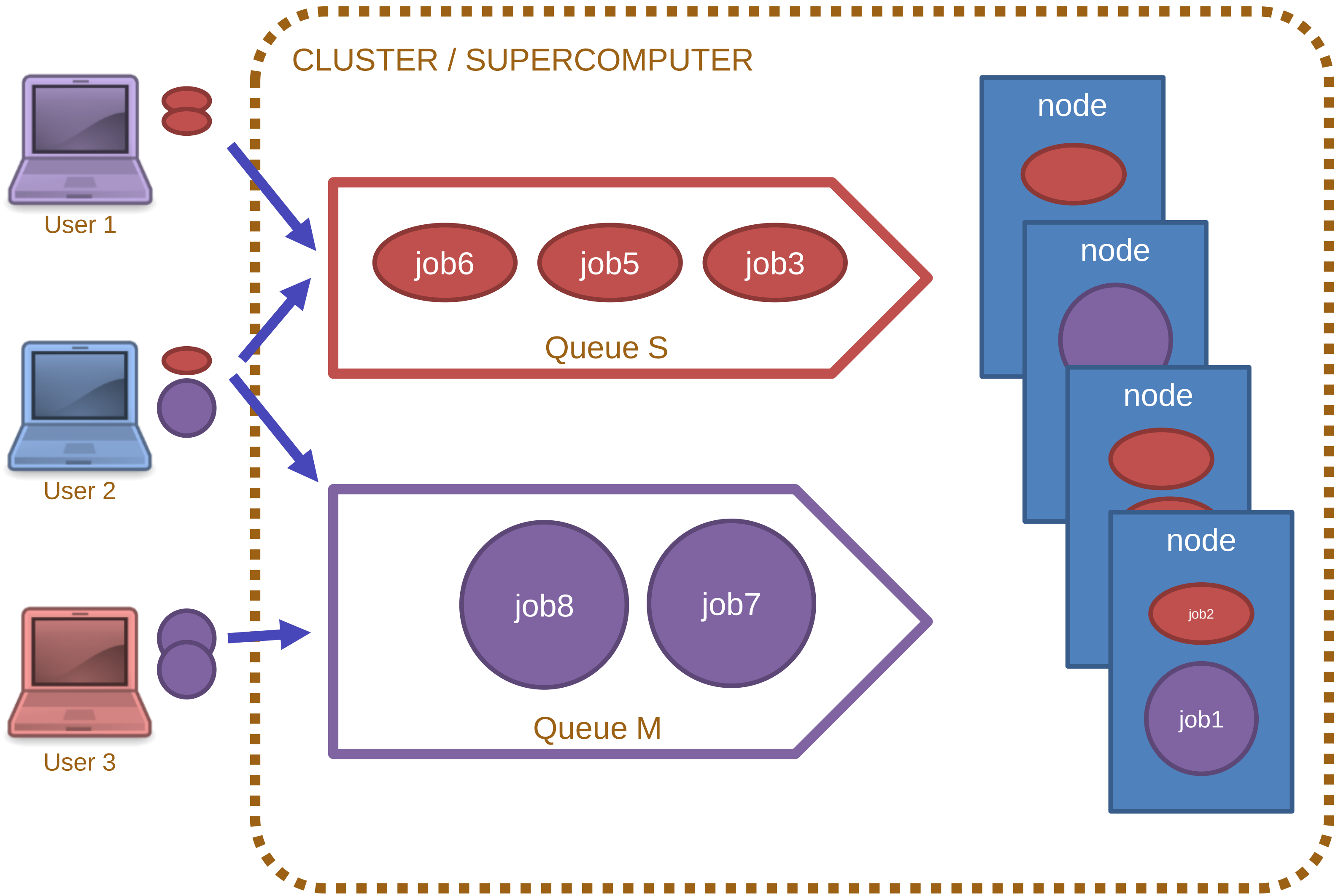


Image from https://alces-flight.com

- Batch scheduling is like playing Tetris – only you can't rotate pieces (says Gonzalo Rodrigo)

Scheduling usually relies on queues



What clusters and HPC systems are good for:

- Executing many tasks at once
  - Use scheduling algorithms
- Serving many users at once
  - Use fair-share algorithms
- Executing tasks that need massive resources
  - Processing capacity, but also memory
  - Word of the day: Exascale
    - » Countries compete to build ExaFLOP-capacity machines

What if we need many clusters/HPCs?

- Not everybody has an ExaFLOP machine
- Scientists often have access to several clusters and HPC systems
  - Different accounts
  - Different passwords
  - Even different operating systems
  - And different sysadmins!
- Solution: equip every system with a dedicated service that will:
  - Work with digital identities
  - Adapt your workload to local system peculiarities
    - » And make sure the system has the software you need
  - Facilitate access to data wherever they are
- This solution is called **Grid**

Some Grid precursors

Distributed file systems: AFS, NFS4

- First implementation in ~1984
- Allow different systems to have common storage and software environment

Condor/HTCondor pools

- High Throughput Computing across different computers
- Started in ~1988 by pooling Windows PCs
- A variant often used as a cluster batch system

Networked batch systems: LSF, SGE

- Could use single batch system on many clusters since ~1994
- Variants of regular cluster batch systems

Volunteer computing: SETI@HOME, BOINC

- Target PC owners since ~1999
- Supports only a pre-defined set of applications

Grid concept – formulated in ~1999

Abstracted interfaces from systems
- No need for common batch systems or common file systems

Introduced security infrastructure
- Single sign-on
- Certificate-based authentication and authorisation

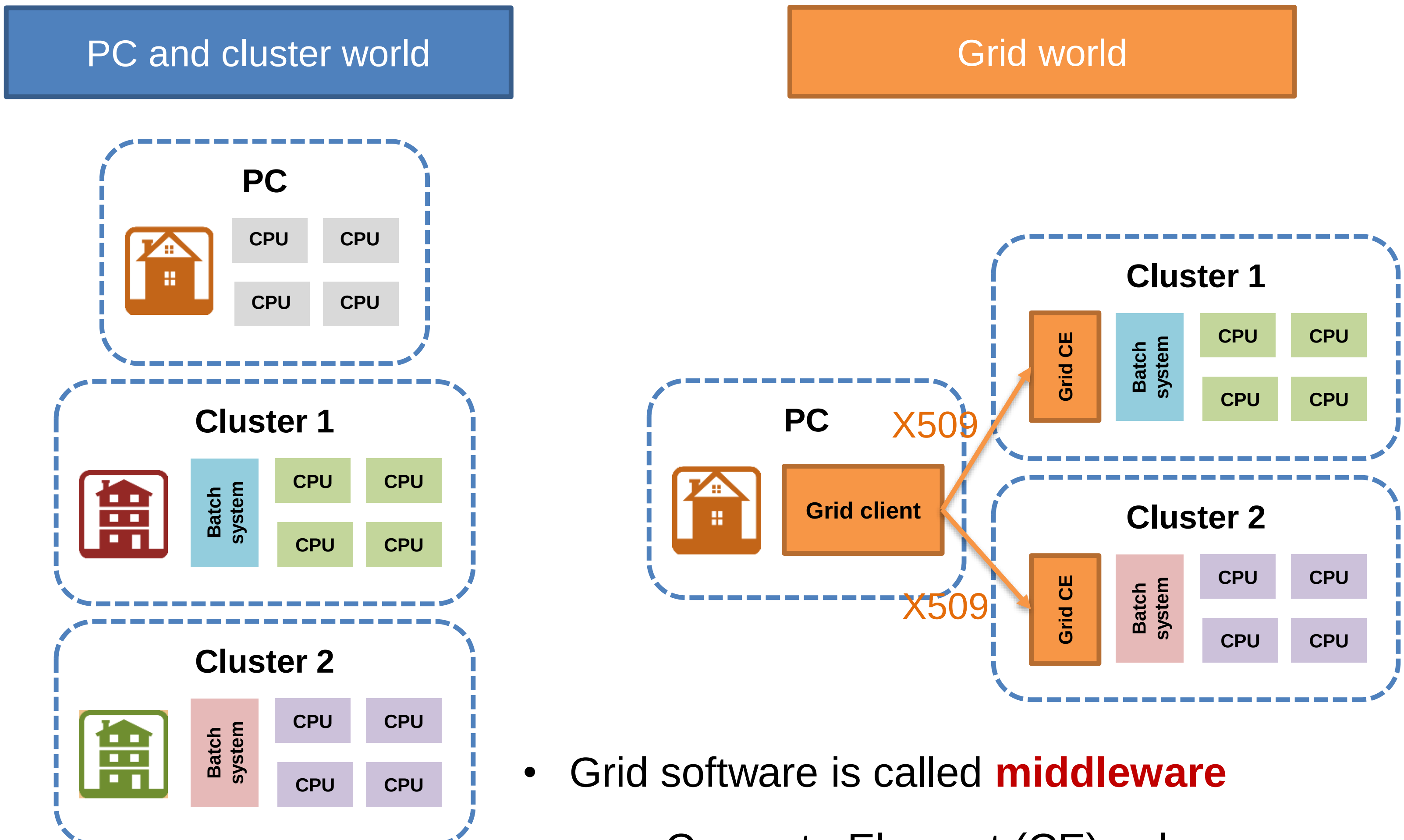
Introduced resource information system
- Necessary for batch-like job management

Ideal for distributed serial jobs
- Initially was thought to be suitable even for parallel jobs

Grid is a technology enabling federations of heterogeneous conventional systems, facilitated by fast networks and a software layer that provides single sign-on and delegation of access rights through common interfaces for basic services

LUND UNIVERSITY

Grid as abstraction layer for computing



- Grid software is called **middleware**
  - Compute Element (CE): a layer between the system and applications

Workflow: Grid vs PC/cluster

PC/cluster

- **Log in via SSH**
  - Different logins on different machines
- **Familiarize with the environment**
  - OS, installed software, storage space, batch system, sysadmin etc
- **Customize the environment**
  - Pathes, environment variables, own software, scripts, data files etc
- **Prepare for batch submission**
  - Interactive execution of short jobs, optimization of batch scripts
- **Submit jobs to the batch system, check their status**
  - Different batch systems (or none) on different machines
- **Log out**
- **Log in later to fetch the output**

Grid

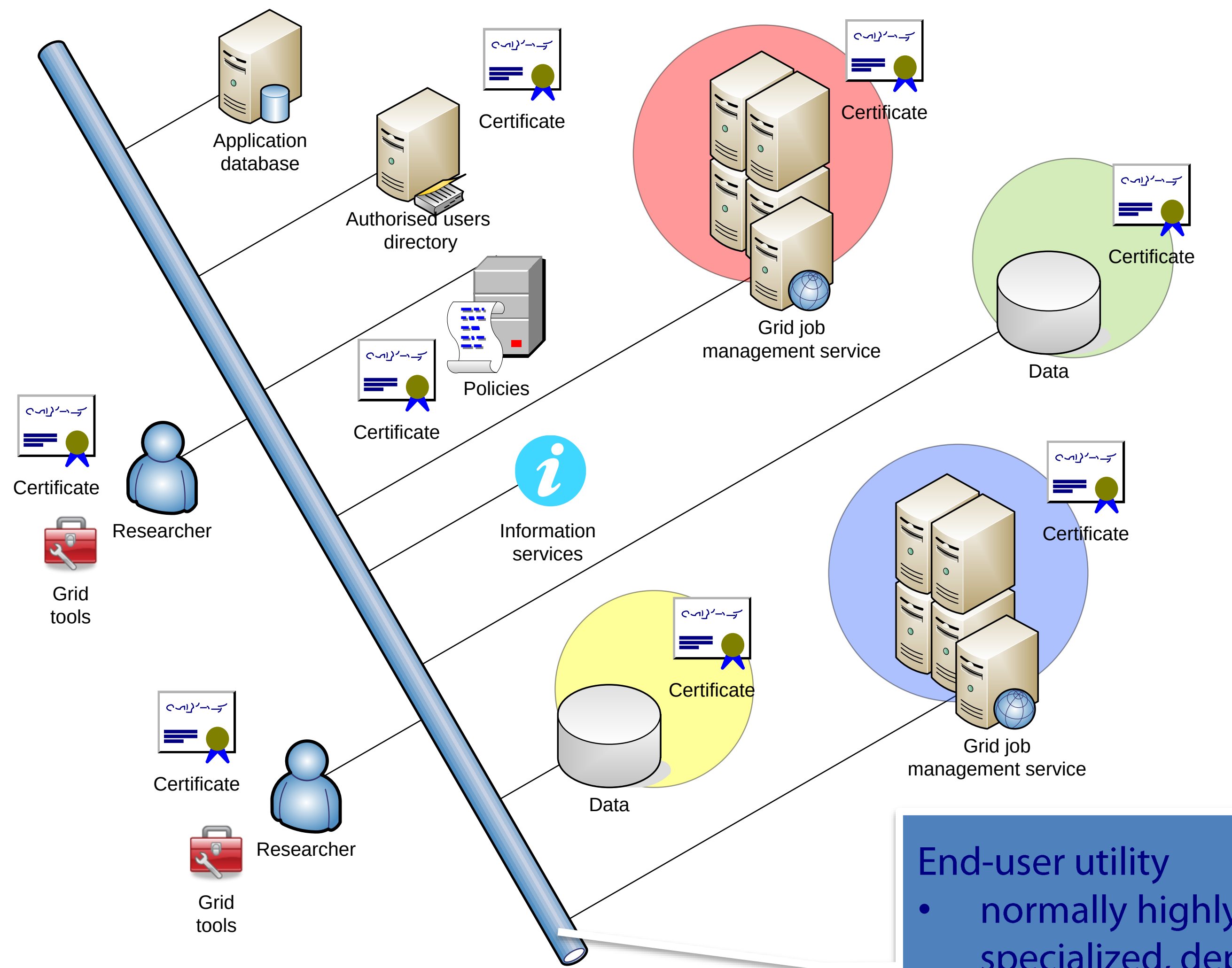
- **Create proxy**
  - One for all machines
- **Create a Grid job description document**
  - Generalization of batch scripts, plus input/output data location etc
- **Test a couple of jobs, fix job description**
- **Submit jobs to the Grid, check their status**
  - Same commands for all machines
- **Watch output appearing in the desired location**
  - Or fetch it manually

Key differences

Operation	PC/Cluster	Grid
Log in	Interactive SSH session	No actual log in: delegation
	Different passwords	Single sign-on
Job description	Shell script with batch-system-specific variables	Specialized language
	Different scripts for different batch systems	Same document for all systems
Environment	Can be personalized	Pre-defined, generic
	Can be explored in detail	All details can not be known
Job monitoring and management	Requires log in	Remote
Data management	Manual	Can be automatic

LUND UNIVERSITY

Overview of generic Grid components



End-user utility

- normally highly specialized, depending on the research task

LUND UNIVERSITY

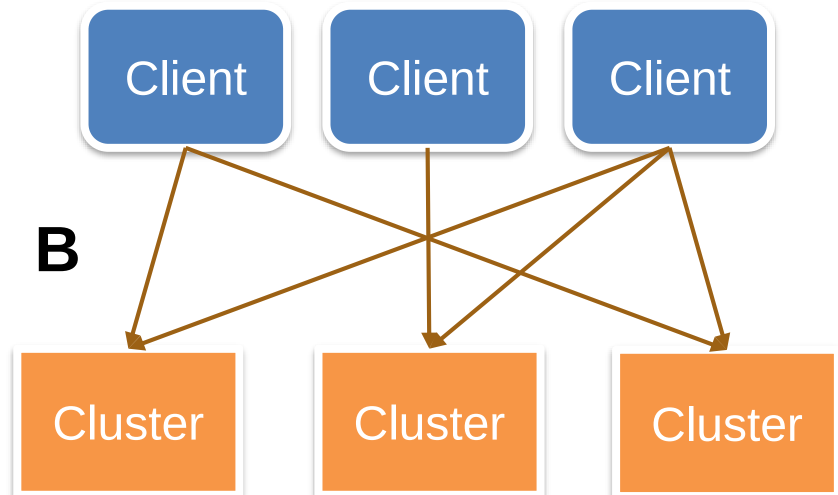
The core of the Grid: Computing Service

- Once you got the certificate (and joined a Virtual Organisation), you can use Grid services
- Grid is primarily a distributed **computing** technology
  - It is particularly useful when **data** is distributed
- The main goal of Grid is to provide a layer for:
  - Common authorization, **single sign-on** – by means of proxies
  - Common task specification (**job description**)
  - Common protocols and interfaces for job and data management
  - Common accounting and monitoring
- All this is provided by Grid **Computing Services**
  - A single instance of such service is called a **Computing Element** (CE)
    - » You also need a Grid **client** software to communicate to Grid services (CE, storage etc)
    - » More during the last class

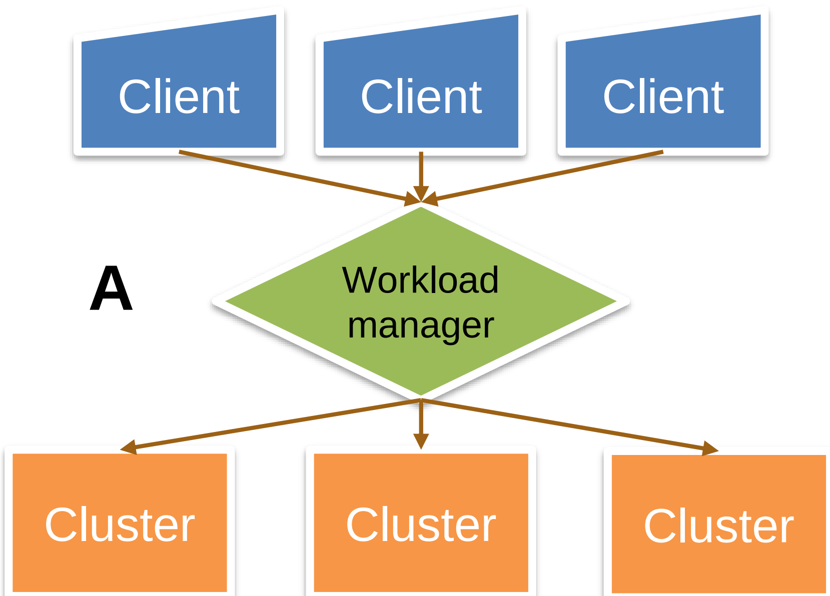
Grid workload management concepts

- Traditional approach **A**:
  - One central service to orchestrate the workload
  - Meta-queue on top of other queues
- Problems:
  - Limited scalability
  - Single point of failure

B



A



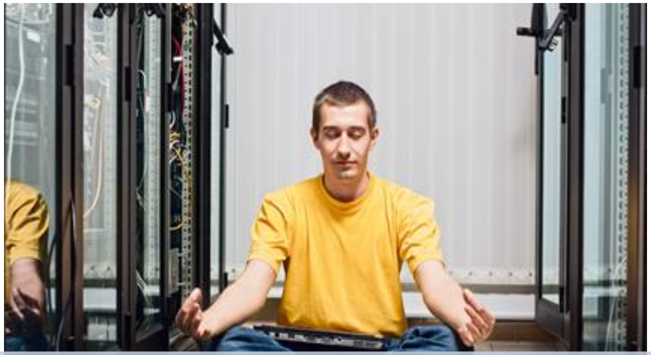
- Alternative approach **B**:
  - Every client can submit jobs to any cluster
  - No single point of failure
- Problems:
  - Non-optimal workload
  - Rather complex clients
  - Slow interaction with users

LUND UNIVERSITY

What's wrong with clusters / HPCs / Grids?

Authentication required (typically, SSH)

Batch queues – interactive work discouraged



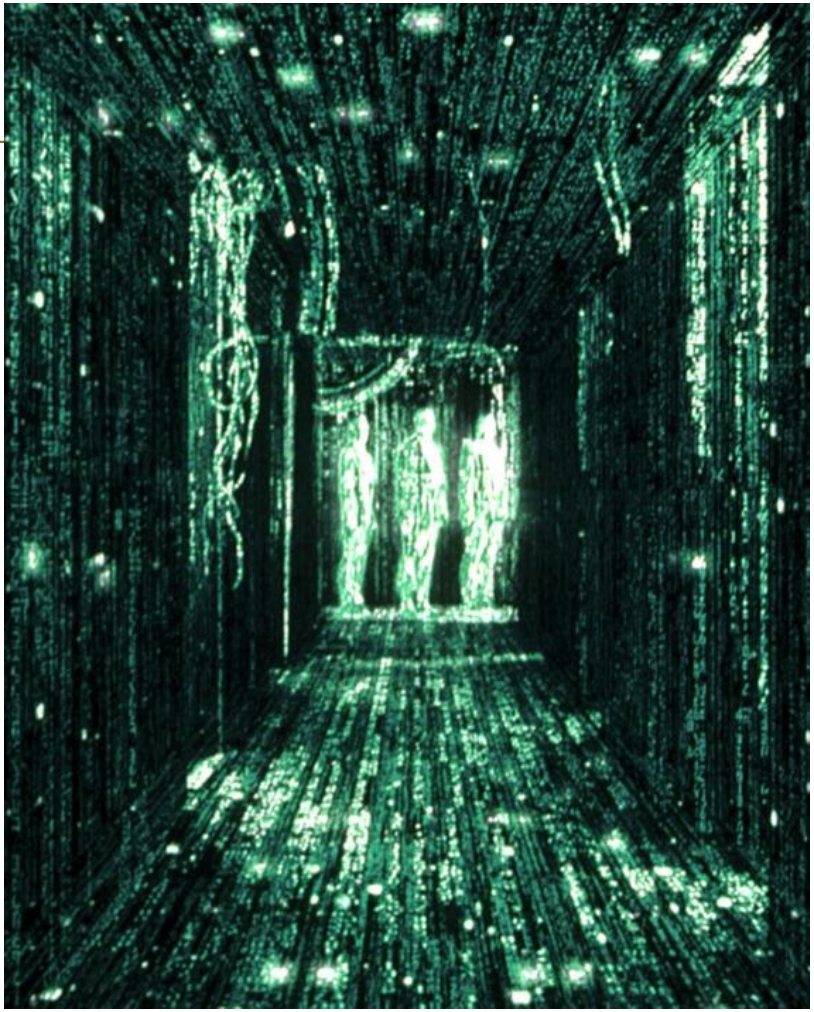
Only sysadmins can really install software

Applications need to be tailored for each site

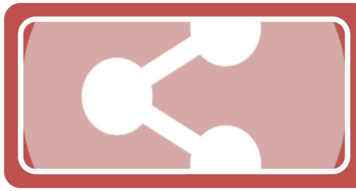
LUND UNIVERSITY

Enter Clouds

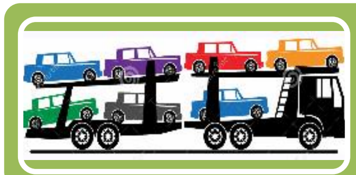
- Cloud is like Matrix for computers
  - You can build your own virtual PC with virtual CPU, virtual memory, virtual network, virtual everything, with your real application inside
  - You can save all this in one file – "image"
  - Load this image into the ~~Matrix~~ Cloud and enjoy instant computing power!
  - And you can even build your own virtual cluster where you are the sysadmin
    - » If you have enough money



Practical motivations for Clouds



Easy to create a platform for shared access – like e.g. Google Docs



Each real machine can host several virtual ones – “**elasticity**”

Convenient way of offering different operating systems on the same hardware, like e.g. in universities

Preservation of software and data: save a snapshot of an old system, no need to re-build for new ones

Recovery and migration of services: when a virtual server crashes, it is trivial to restart it on a healthy host

Isn't Cloud another word for Grid?

- Yes, by inference: Grid is inspired by BOINC, BOINC requires virtualisation, virtualisation is used by Cloud
  - Jokes aside, both Grid and Cloud are technologies to abstract underlying infrastructures
- No: Grid is Communism, Cloud is Capitalism
  - Clouds typically have clear ownership, Grids are federated infrastructures
- Maybe: virtual clusters become common in Grids
  - Infrastructure as a Service
  - More in the next part!

Virtualization and Containers

Florido Paganelli
Lund University
florido.paganelli@hep.lu.se

General-purpose computing: Theory

1)Write code

2)Compile on ANY target platform

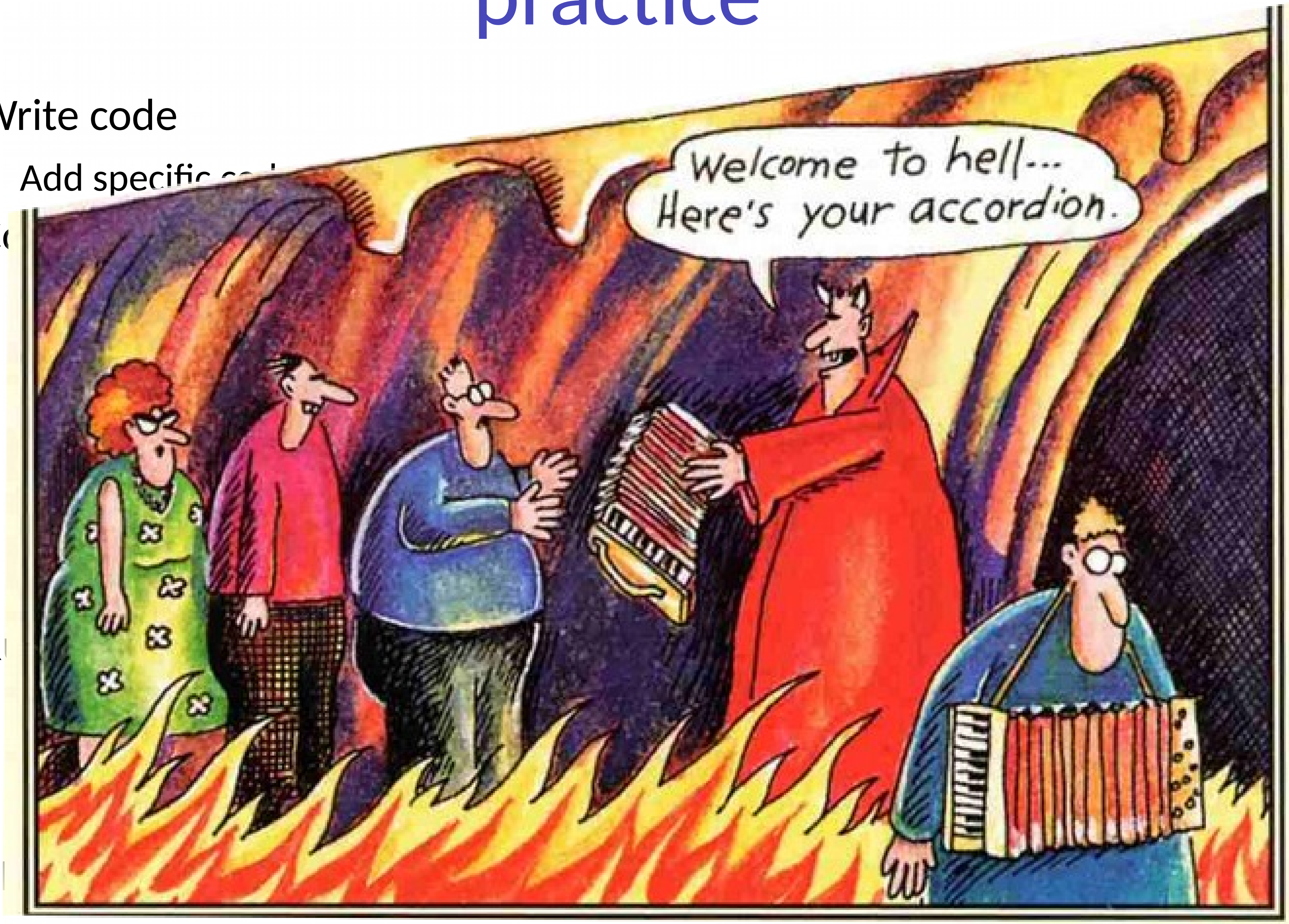
3)Run on ANY target platform

General-purpose computing: practice

1. Write code
   1. Add specific code to adapt to each platform
2. Compile on target platform
   1. Install **all required development libraries** on target platform
      a. decide: binaries (dynamic) or sources (static)?
   2. Setup the **toolchain**: instruct your build system about which tools to use and where to get them
   3. Setup the **build environment**: instruct your build system about which tools to use and where to get them
   4. Write a script to do 2.1, 2.2, 2.3 automatically
3. Run on target platform
   1. Install **runtime dependencies**
   2. Instruct the code where to find runtime dependencies
   3. Write a script to do 3.1, 3.2 automatically
4. TEST ON EVERY POSSIBLE PLATFORM

General-purpose computing: practice

1. Write code
   1. Add specific c
2. C
   1.
   2. ere
   3. se
   4.
3. R
   1.
   2.
   3.
4. T



Setting up the build & runtime env

- No real unified way.
- Experiments have their own scripts/packages/suite/tools/ whatever
  - tools to create/discover toolchains and envs (autotools, cmake)
- CERN: `cvmfs` or shared filesystems with custom scripts
- In HPC/HTC, DIY or ask sysadmin
  - Software to build development and runtime libraries (`EasyBuild`)
  - modules (module or lmod) to load an unload environments on demand

Packaging. A wish

- What if I could package my software all together with the libraries to create and run it?

Packaging. You wish...

CERN software for ATLAS:

- Custom system packages required for each operating system
- The most skilled sysadmins in Umeå tried 3 years ago to compile all software and gave up after 2 years. Continuous updates and new libraries where the main issue

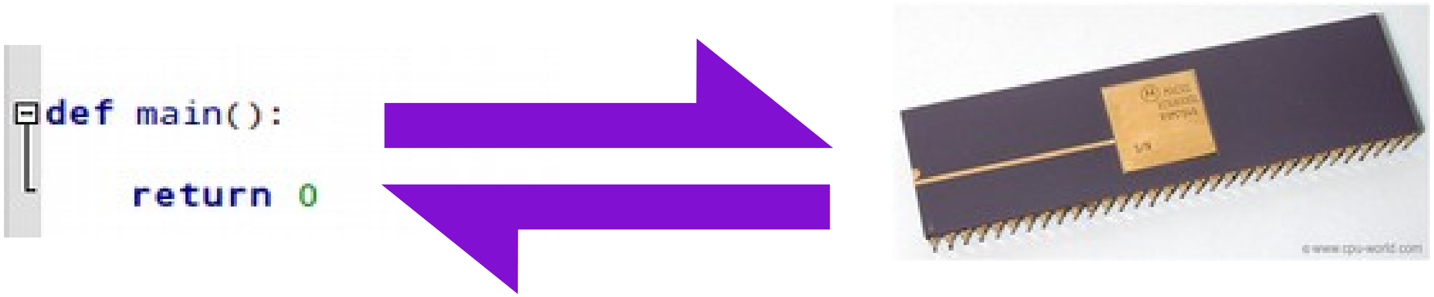
Packaging. Another wish

- ~~What if I could package my software all together with the libraries to create and run it?~~
- Ok what if I send around my preconfigured computer

Send around a machine

- It’s clear that shipping hardware around the world is not the solution:
  - It might get lost
  - someone might compromise it on the way
  - There exist only one copy of it, unless you buy bunches. Bit expensive.
- Solution: have a “software” representation of a machine. Is that possible?

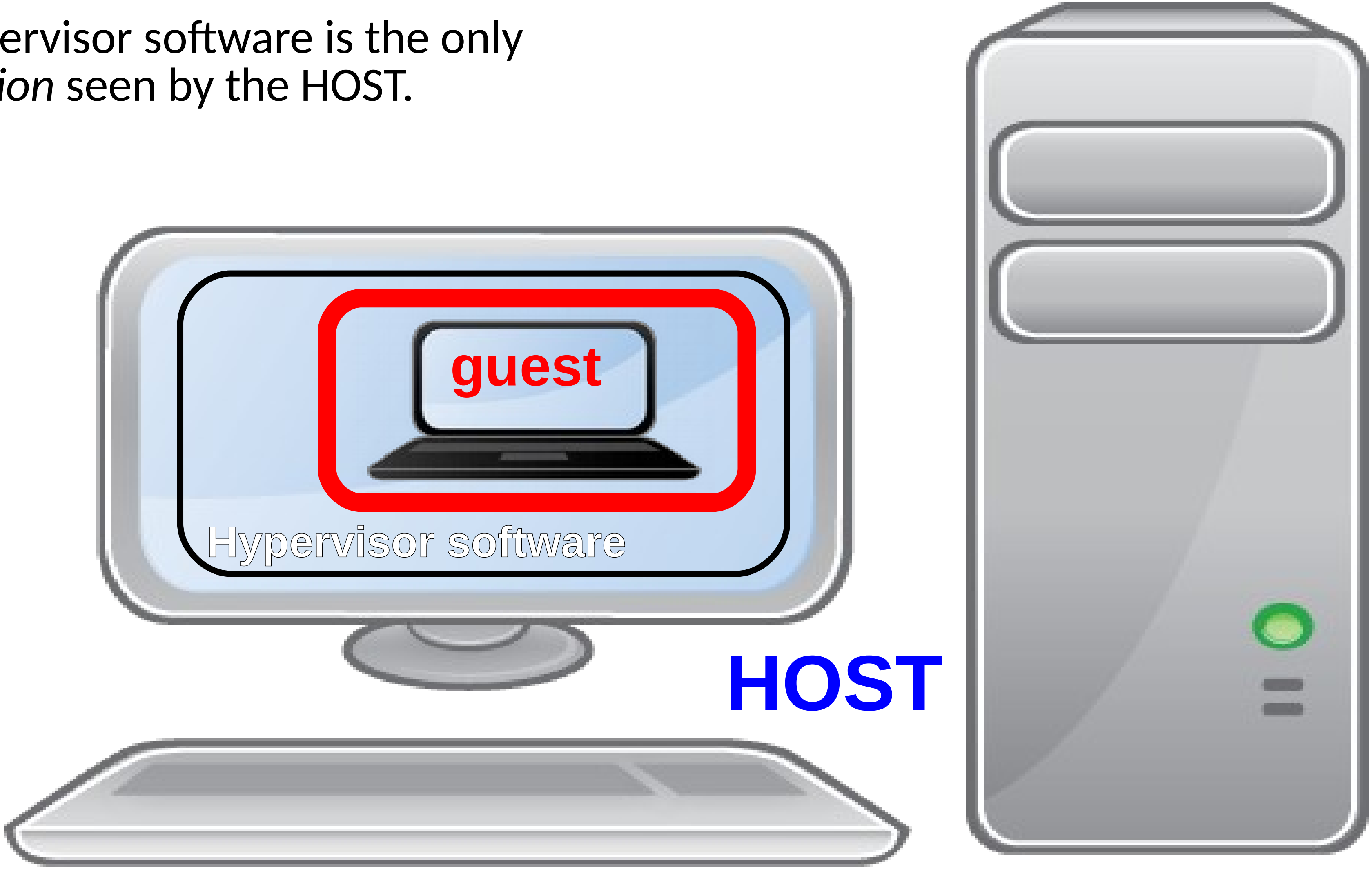
Hardware-Software equivalence



- Everything that can be modeled via software can be created in hardware and vice-versa
- This poses the foundation for machine (hardware) **simulation** and **emulation**
  - **Simulation**: software that behaves exactly like some piece of hardware, internally and externally. For prototypes and testing
  - **Emulation**: software whose *external* behavior *acts as a* piece of hardware. The internals can differ. It ”pretends” to be some hardware.

Virtualization

- Running a virtual computer (guest) inside a physical computer (host)
- The Hypervisor Software *emulates* real hardware
- The Hypervisor software is the only *application* seen by the HOST.

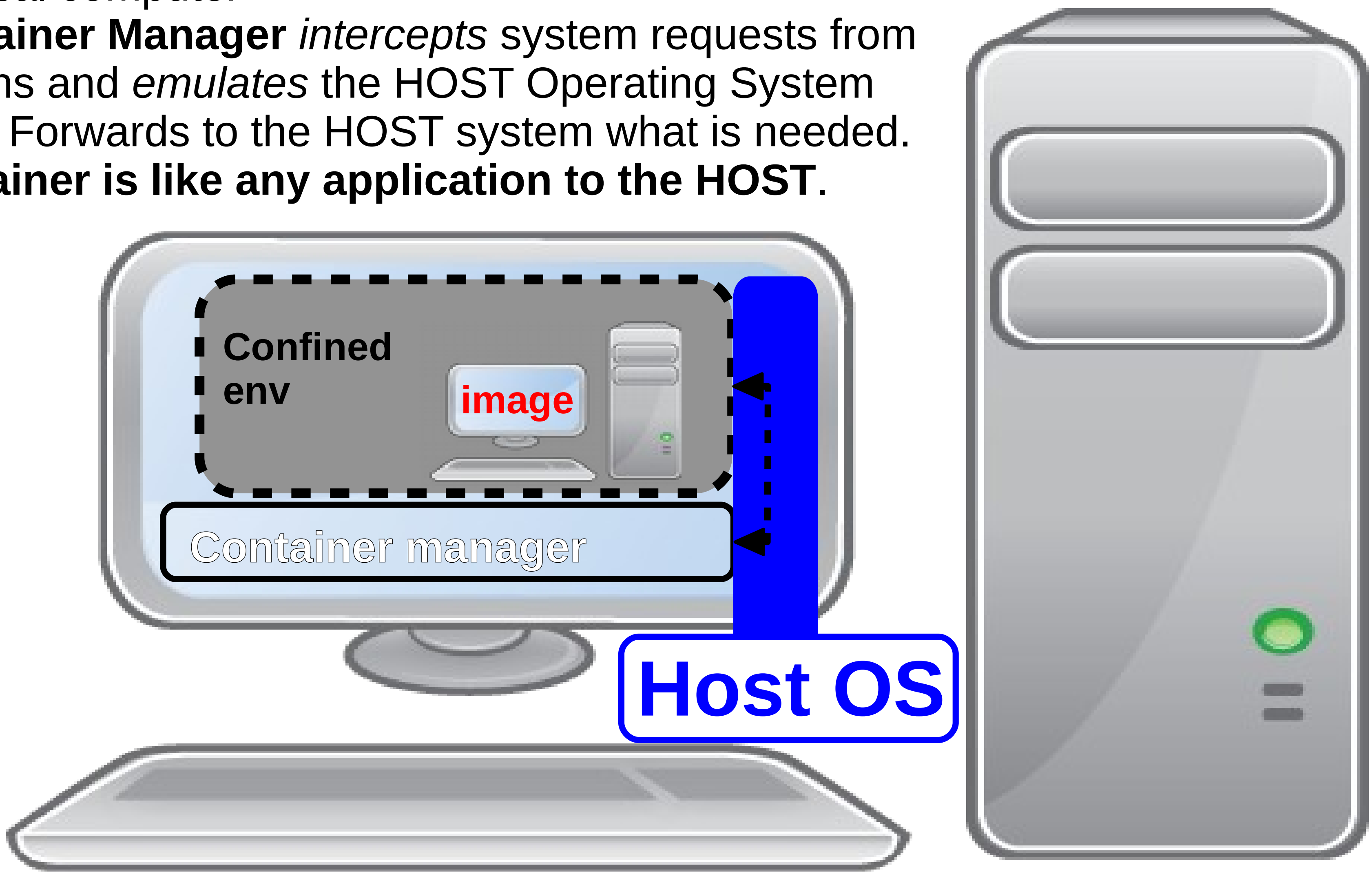


Virtualization

- Running a computer (guest) inside a computer (host)
- The guest machine is usually called Virtual Machine. It carries a full representation of an hardware configuration.
- The Host machine manages the guest machine using something called Hypervisor. The user can manage the machine using the tools provided by the hypervisor.
- The host offers software simulated or emulated hardware, plus it can offer **real** hardware to the guest machine
- The guest machine sees all the software simulated/emulated/virtualized hardware as it was real hardware, but it can also be aware that it is virtualized to boost performance
- A virtual machine can contain ANY operating system regardless of the Host OS. (Windows, Linux, Mac, BSD, Android?)
- Was invented mostly for
  - Ease of management of multiple machines in big data centers, without the need to buy dedicated hardware for each machine
  - Emulate hardware and run multiple operating systems on the same machine

Containerization

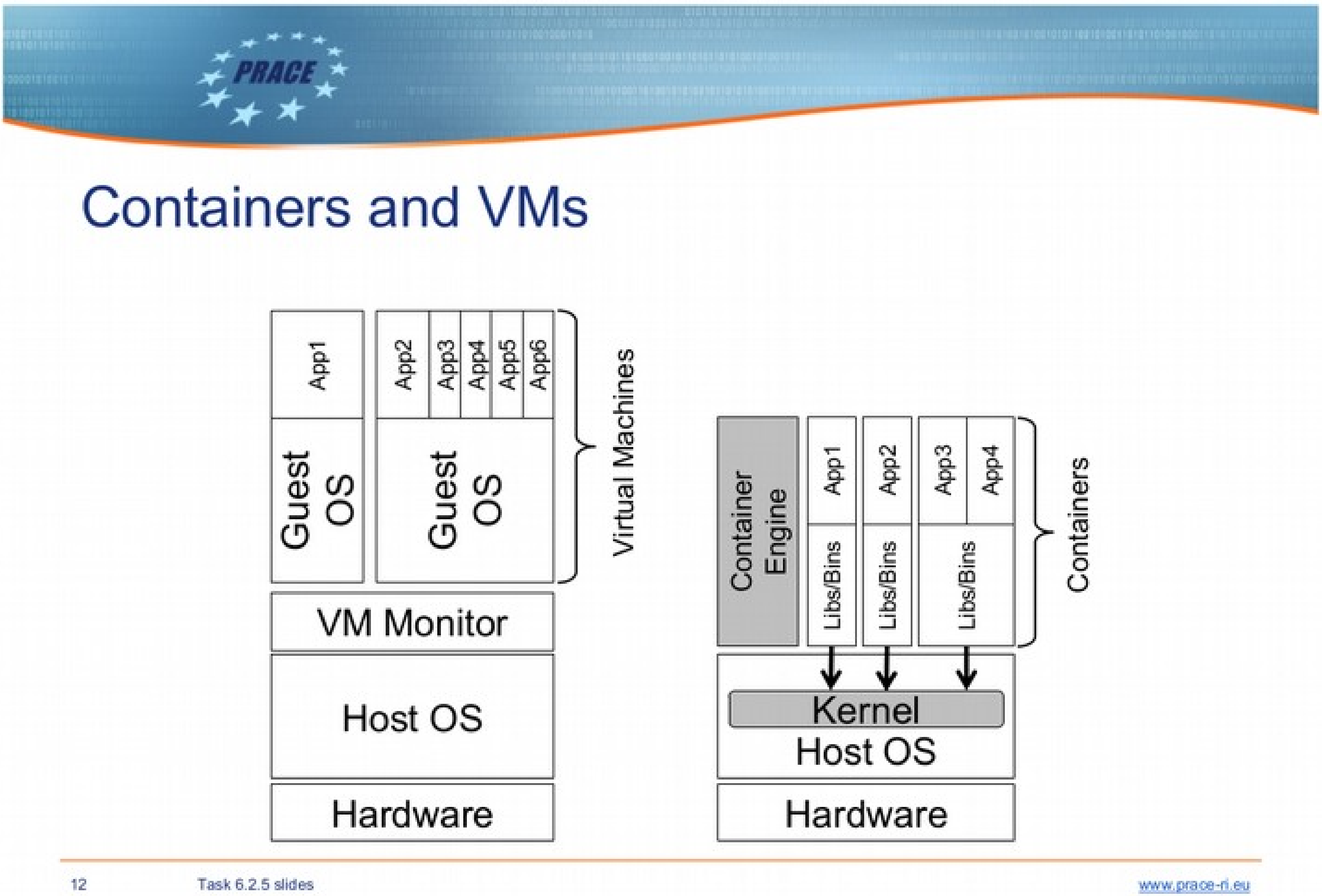
- Running a different* Operating System **image** inside a *confined environment* inside the HOST Operating System of a physical computer
- The **Container Manager** *intercepts* system requests from applications and *emulates* the HOST Operating System response. Forwards to the HOST system what is needed.
- **The container is like any application to the HOST**.



Containerization

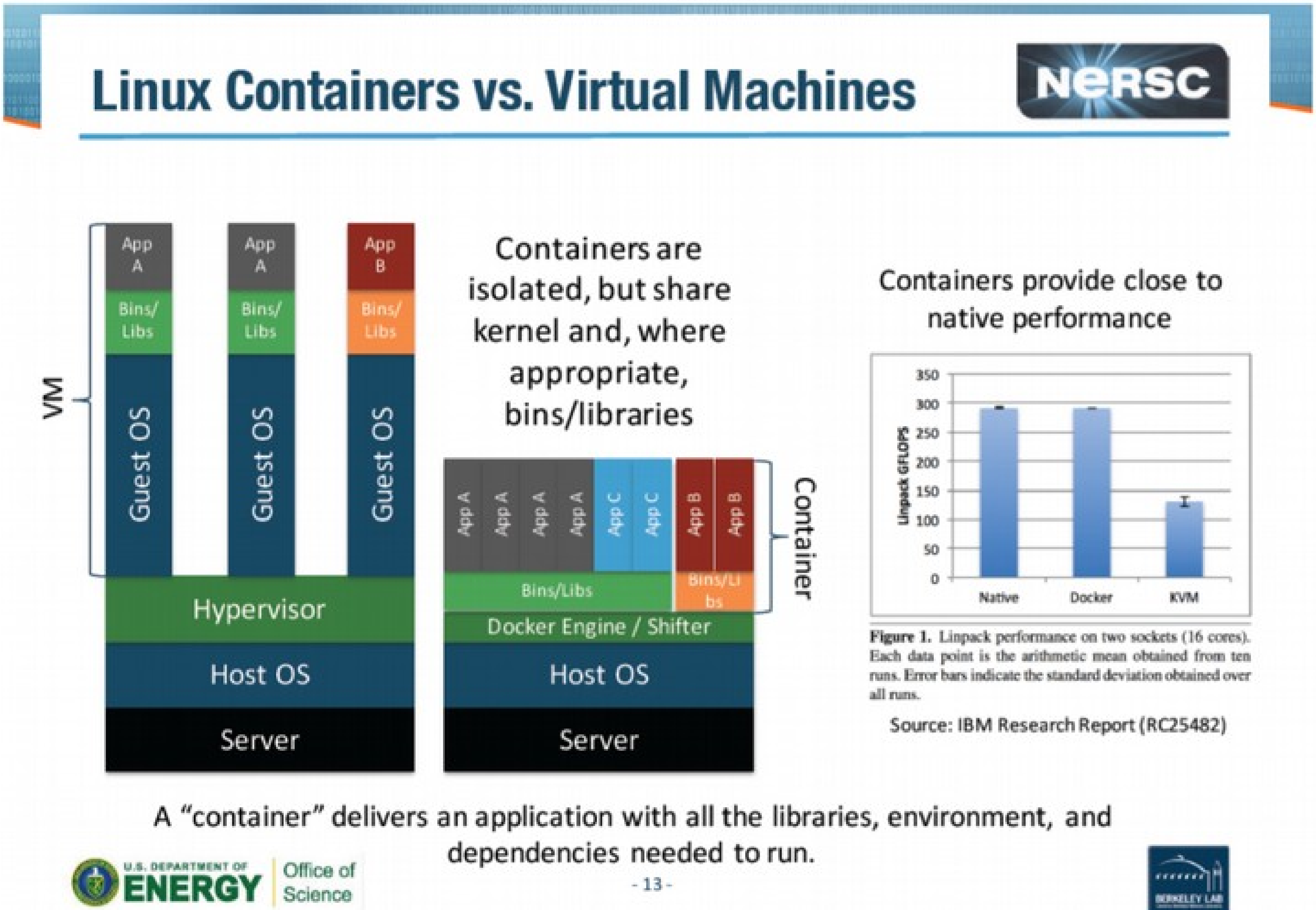
- Running an Operating System **Image** inside a **confined environment** in an existing host operating system)
  - *Different OS: Not all operating systems can work in a container. Only those that share a similar *kernel architecture*, that is,
    only Linux machines on a Linux Host OS.
    In case of a different kernel, the kernel requests are *translated* to the Host Kernel.
  - **Image**: a “snapshot” of the files in a system
- The user can manage the **Image** using something called Container manager, a set of tools and commands to deploy/start/stop/install/configure the contents of the image.
- The Host OS treats the container and its applications like a single application running in the OS. The OS activity inside the container is *emulated* by the container system.
- The application in the Image sees only the containerized OS and NOT the Host OS. It sees the exact same hardware that the Host OS has.

Architecture summary



Credit: Abdulrahman AZAB, PRACE technical talk, NorduGrid 2018, Garching Bei München, Germany
http://indico.lucas.lu.se/event/908/session/3/contribution/12

Architecture facts



Credit: Abdulrahman AZAB, PRACE technical talk, NorduGrid 2018, Garching Bei München, Germany
http://indico.lucas.lu.se/event/908/session/3/contribution/12

When to use what?

Virtualization:

- Environments in which you have total control
- Environments that must run a different operating system technology within a HOST OS (Example: running a Windows VM inside a Linux HOST)
- Complex setups that involve different platforms (OS technology, hardware architecture)

Containers:

- Systems with similar OS technology (i.e. multiple linux versions/distros)
- Application confinement
- Complex test setups that involve multiple distributed systems

“a ghost is haunting computing”

- Virtualization and containerization was so cool that hardware vendors included the so-called virtualization instructions
- A VM/Container is allowed to *see and use* the actual hardware
- 2018: Meltdown and Spectre security vulnerabilities
  - Every processor since 1995 (few exceptions)
  - A process can *leak* its memory info to *some other process*
  - The fixes out there slow down performance of at most 20%
  - Source: https://meltdownattack.com/
- *=> Virtualization and containers are considered intrisically insecure until a new generation of processors is born*
- => It’s hard to convince a sysadmin virtualization is good!

The Cloud

- With a huge computing power, say an entire data center, one can create a large number of virtual machines or containers on demand
  - *Cloud interfaces* to request resources (computing or storage)
    - **IaaS,** Infrastructure as a service : request a computer architecture or a whole network of computers. It only provides the hardware (virtualized or not)
    - **PaaS**, platform as a service: request some hardware and an operative system inside, along with some management software (could be docker for example)
    - **SaaS,** Software as a service: the user just requests a service to use. A common historical example is web hosting, nowadays one can think of services like Jupiter Notebooks servers or Online latex editors like overleaf.
  - Products: OpenStack, Google Kubernetes, Amazon EC2 / S3...
    - **Not compatible with each other, for marketing reasons.**
- Users can create their own predefined setup and make it available on the internet.

Next steps

- In this course: Containerization
  - **Docker**: do-it-yourself container technology with online repositories
  - **Singularity**: HPC-level container technology.
- NOT in this course: Virtualization
  - VirtualBox: desktop Hypervisor
  - See slides from our MNXB01 course if curious http://www.hep.lu.se/courses/MNXB01/

Containerization in Scientific Computing

- Goal: provide a container ready with the environment for your kind of research

1) Create a container on your laptop and test it

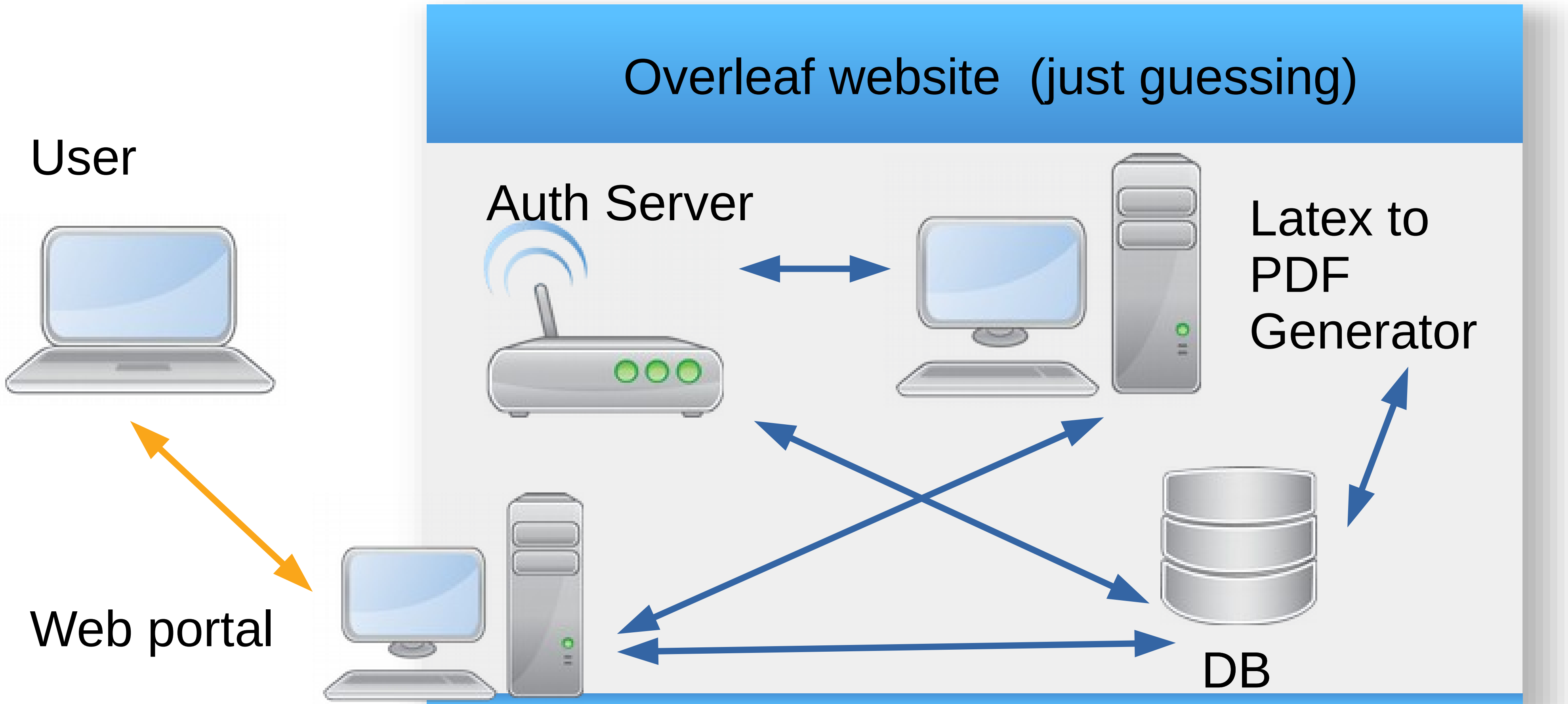
2) Use the container on a research facility (or on the cloud) where that container technology is supported

Technologies: Docker

- https://www.docker.com/: commercial solution
- https://www.docker.com/products/docker-engine community solution that we will use
- Installation and configuration (Will do on the 23rd): https://hub.docker.com/search/?type=edition&offering=community
- Tutorials (TODO: evaluate one):
  - https://training.play-with-docker.com/beginner-linux/ Homework! More info via email.
  - https://docs.docker.com/get-started/ Will do on the 23rd

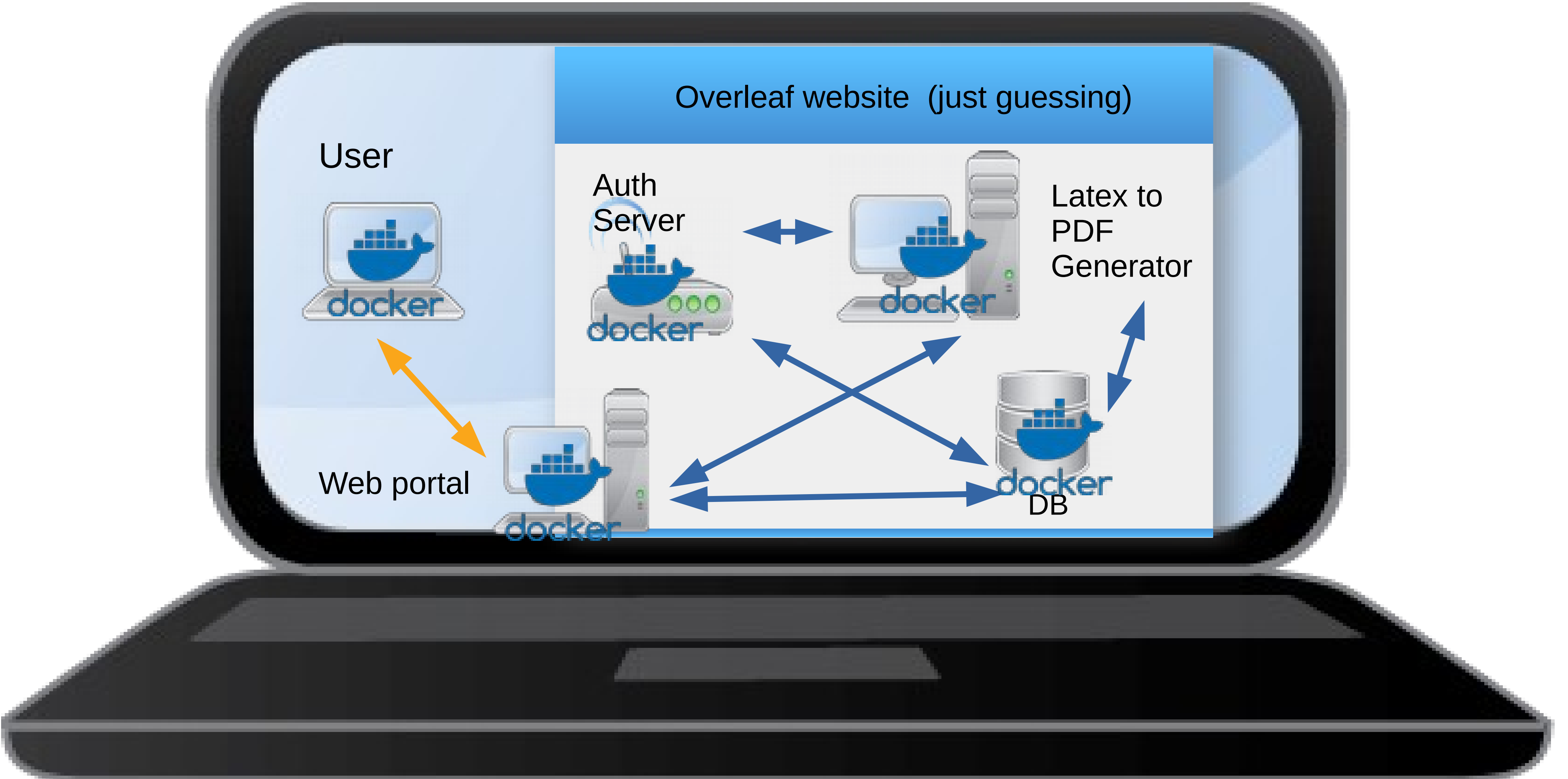
Docker

- Was invented mostly for complex applications that involve different servers (typically, web applications)



Docker

- So that the developer can have everything in their machine to test
- Arrows: docker's own virtual (emulated) network



How will we use docker

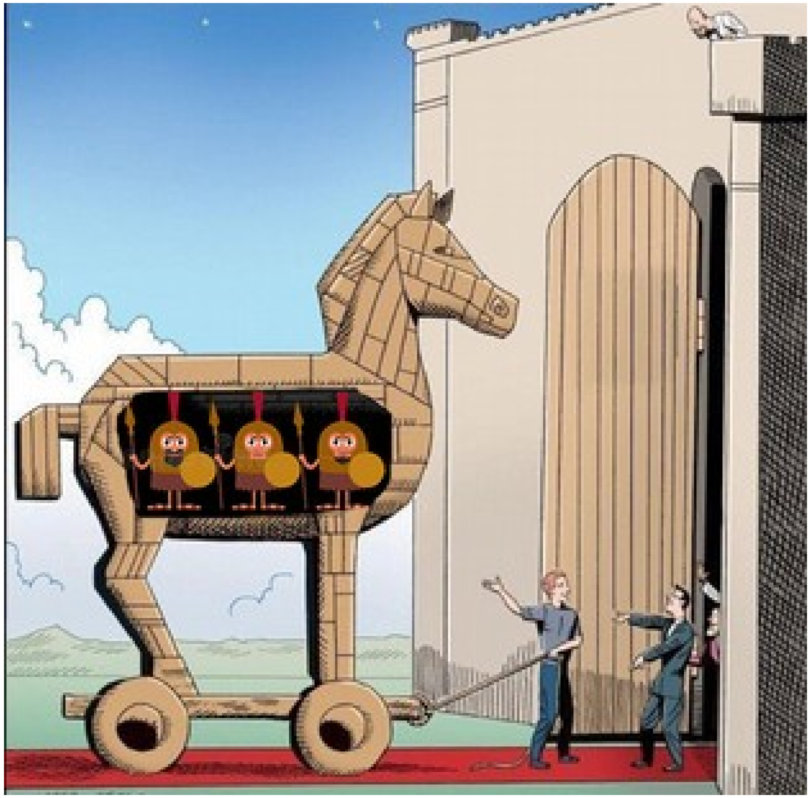
- Create an environment that runs your application
- Give it to other researchers to reproduce your results or to do their research in a similar environment

Problems: security

- Trojan-horse potential
- Datacenters and sysadmins do not like root users
  - An unskilled user might harm the whole system by mistake!

=> **in Datacenters** :

- Images have to be **certified** to be not harmful
- User **cannot be root**
- Images are **readonly**



Implications

- It's not possible to modify the content of the image
- It's not possible to get data out of the image
- It's not easy to install new software in the image

An alternative for HPC: Singularity

- Runs a container in user space
- Mounts local folder inside the container transparently
- Security can be fine-tuned by the sysadmin
- Question: Docker-Singularity compatibility?

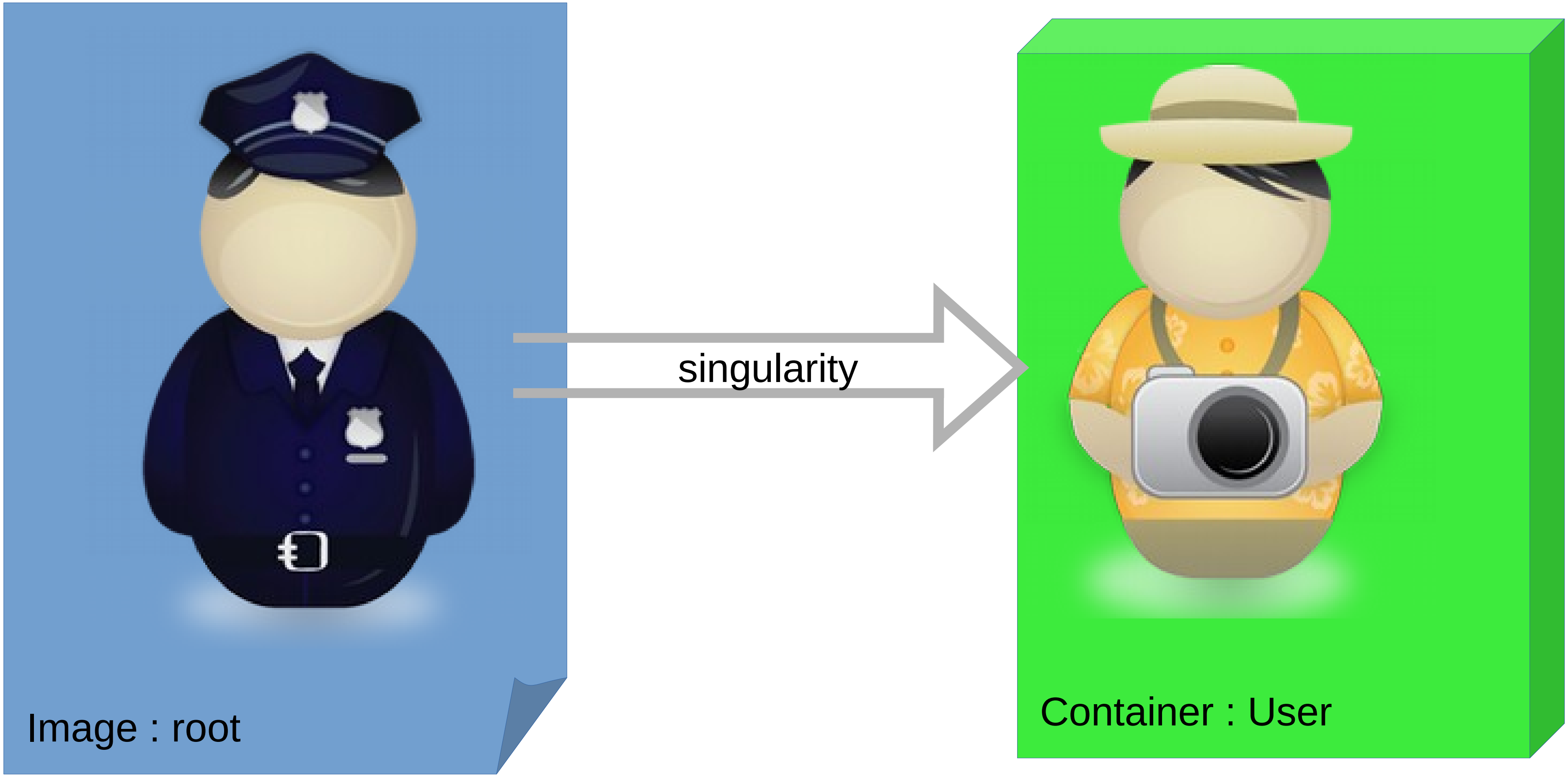
Technologies: Singularity

- https://www.sylabs.io/docs/
- Can be installed by user, but
  we will not discuss this in this course
- A container it’s a file, that is also
  an executable
- A user can invoke the environment for a job by just
  running the container

```
./singularityimage myscript.py
```

Singularity approach

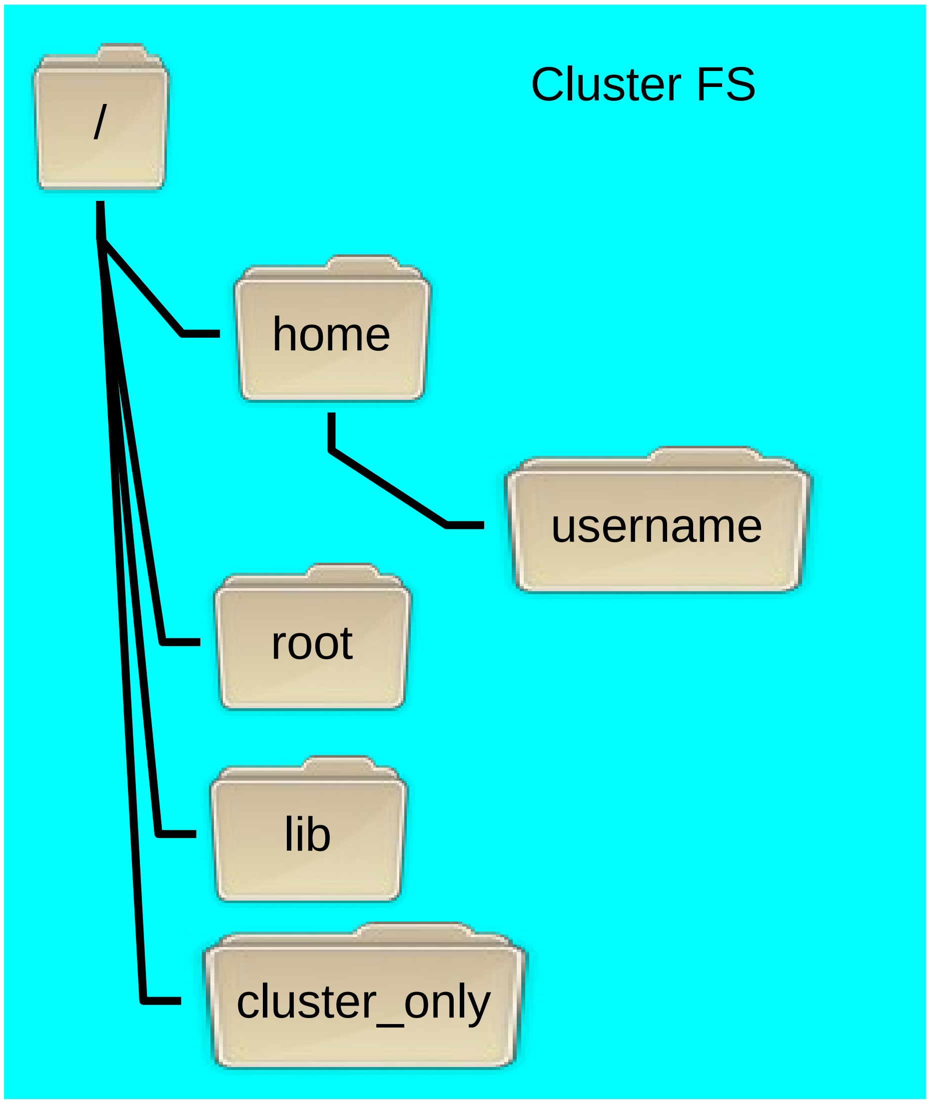
- Root in the image becomes a normal user in the container, hence can't modify the image/container.

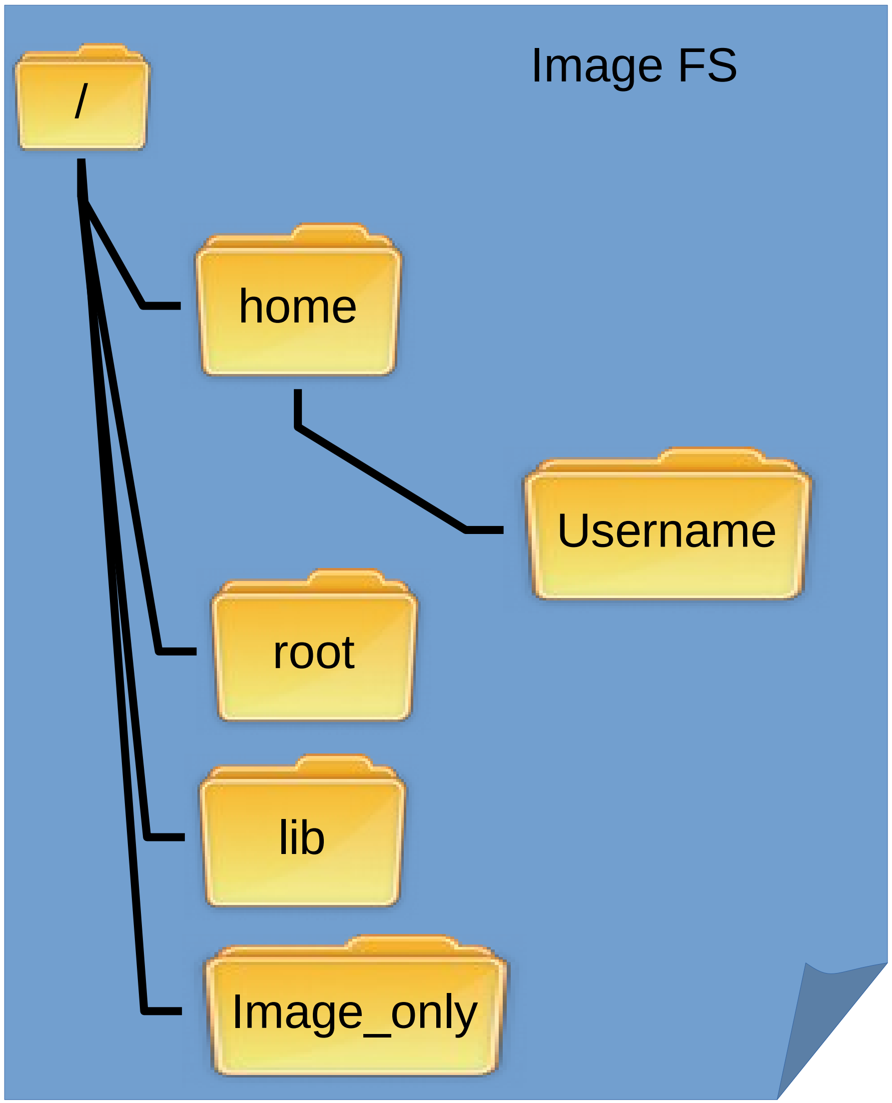


Singularity approach

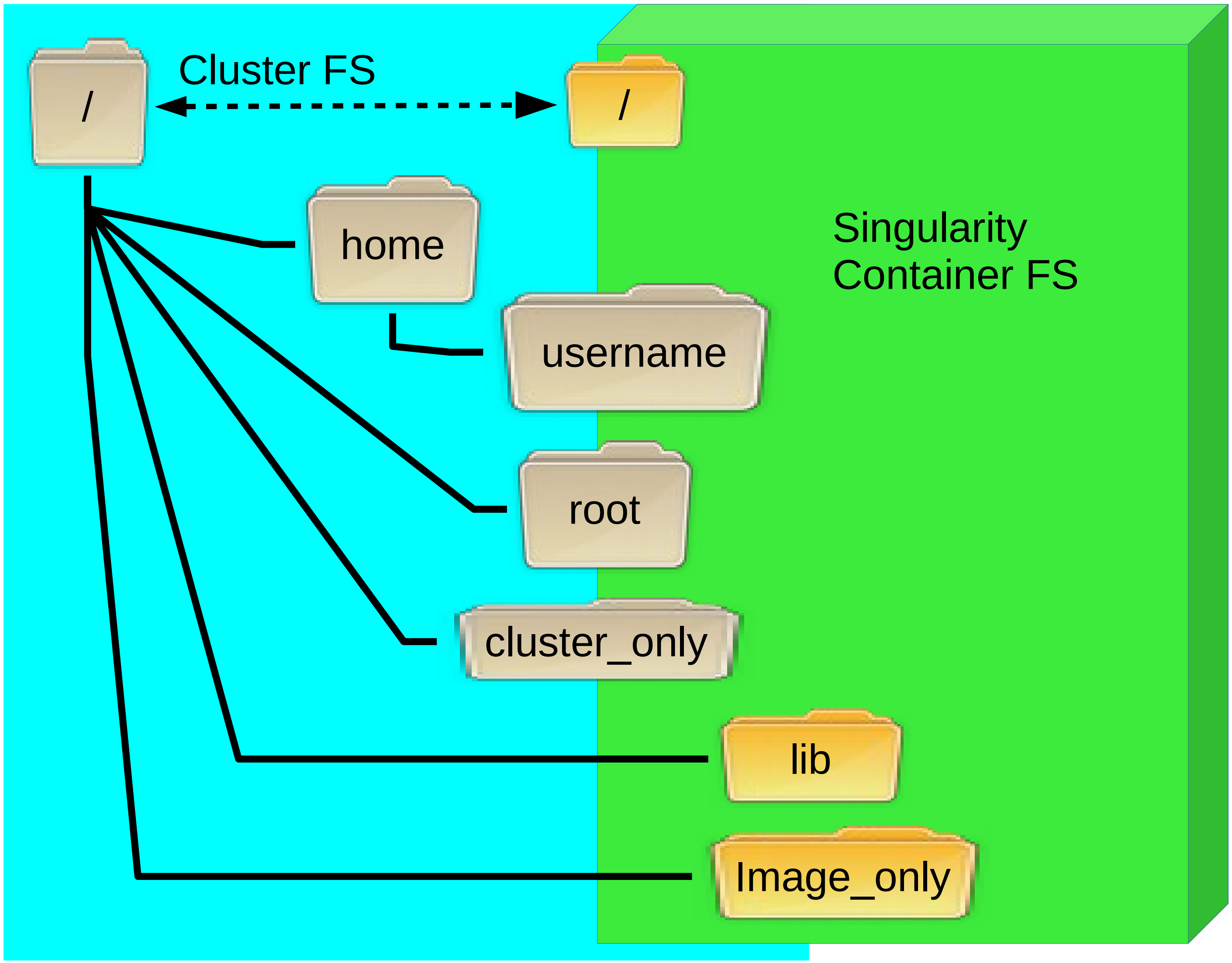
- The user home folders on the cluster are mounted inside the container environment. The user **can write** in their home while inside the container.
- Other paths of the cluster filesystem **that are not present in the image** are automatically mounted in the container.
- Paths in the container that exist on the cluster are **overridden** by the containers one, that is, the user can only see the containers one (overlay effect)

Overlay – before:





Overlay – After:



Prepare for tutorial 18/4

- Completely new to bash command line? Please do this tutorial: https://linuxsurvival.com/linux-tutorial-introduction/
- Need some refresh of linux command line stuff? http://rik.smith-unna.com/command_line_bootcamp/
- Feeling brave? Here’s a tutorial to learn bash scripting. We will use some of this knowledge during 18/4 https://www.learnshell.org/en/Welcome
- Requirement: need to be able to use a text editor on the cluster. We will show you options.

Images references

- Libreoffice Gallery
- Gary Larson, The Far Side, taken from http://www.dedics.co.uk/temp/welcome%20to%20hell.jpg
- https://knowyourmeme.com/photos/1393740-trojan-horse-object-labels
- https://meltdownattack.com/
https://meltdownattack.com/

Software references

- `EasyBuild`, tool to build devlopment libraries
  https://github.com/easybuilders
- `Lmod`, tool to manage build/runtime environments
  https://github.com/TACC/Lmod
- `autotools`, suite to configure build environments for compilation
  https://www.gnu.org/software/automake/
- `cmake`, suite to configure build environments for compilation
  https://cmake.org/