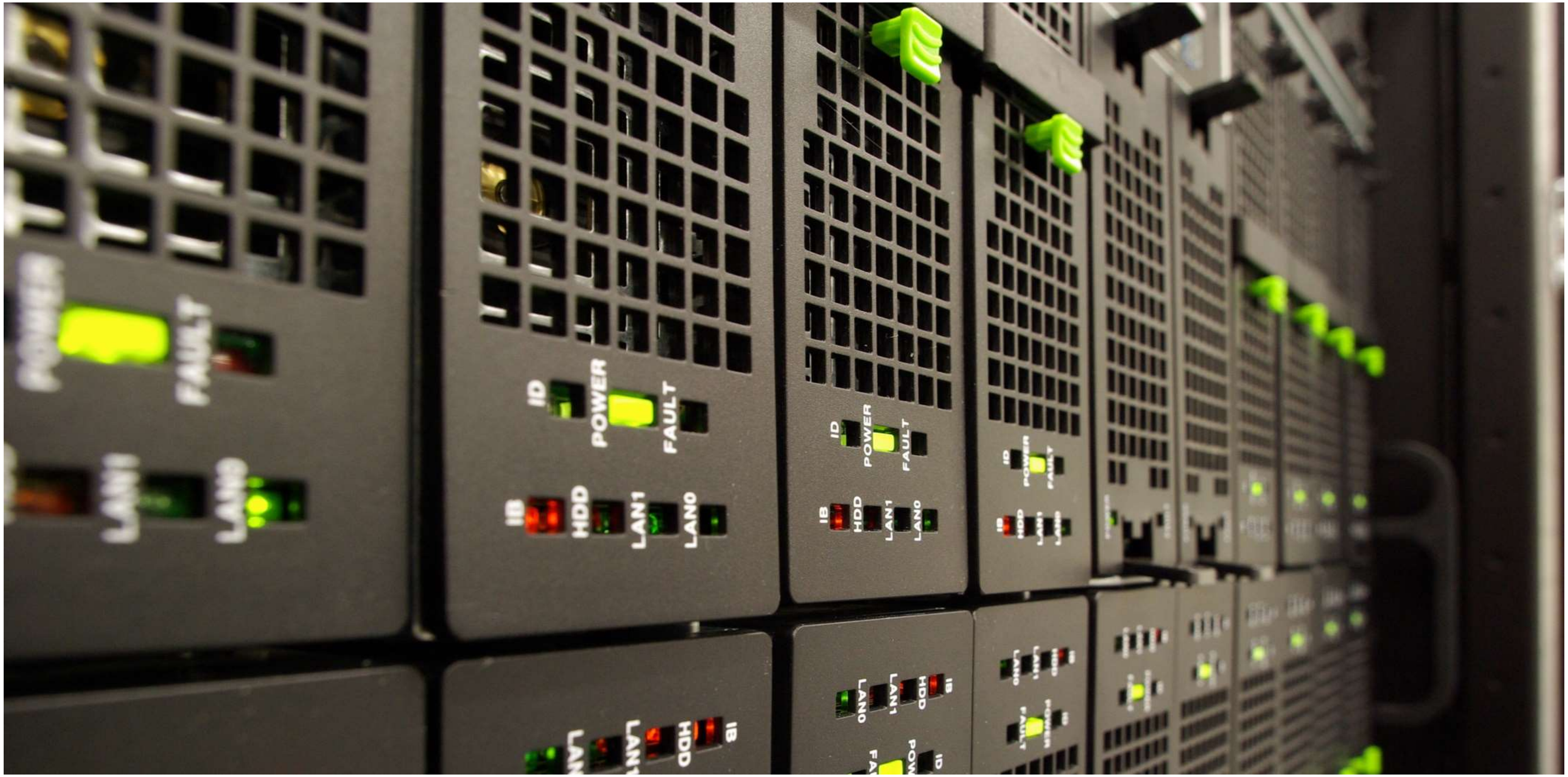

Introduction of Singularity in Aachen

IT Center
RWTH AACHEN UNIVERSITY

History

- Late 2017: First contact with Singularity, first tests and measurements
- 2018: Progress stalled due to other projects (Claix 2018)
- Early 2019: First test cases with software that is hard to get running on CentOS 7
- Works with little effort. Until things get complicated.

Overview

- Singularity 3.4.1 (drawn from EPEL)
  - Pro: Up-to-date version under package control (new features available fast)
  - Con: Singularity‘s development is volatile (things break …)

- Currently: Images provided in a central NFS location
  - Future: Freedom to run images from multiple filesystems
  - Ditch container files for directory-based containers?

- Virtually native performance
  - IB and OmniPath both utilized at native performance
  - MPI fast – if you can provide binary-compatible host MPI

- Global path bindings allow seamless integration of working directories
  - Containerized software has same access as native software
  - No user action necessary: The directories are just there

Test case: Firedrake

- FEM software – developed and tested on Ubuntu

- Not tested on CentOS 7.6
- Performance varies with versions of dependencies (great …)
- Developers provide Docker container with chosen settings and software stack
- Uses MPI for parallel computations

➡ Ideal candidate for testing!

Problems

- Developers create Docker containers for Docker environments, here: Container has user `firedrake` with pre-configured shell and configs

  But in Singularity, we are who we are on the host system! This requires tinkering

- Home directory sharing is both blessing and curse
  - Users can utilize configuration files from host environment - weird side effects can happen

- Leveraging interconnects in HPC-unaware containers requires extra work

- Finding proper host MPI for exotic containers feels like trial & error

What does this look like?

```
Bootstrap: localimage
From: ./firedrake_docker_june.sif

%environment
    export HOME=$HOME
    export WORK=$WORK
    export HPCWORK=$HPCWORK
    export TMP=$TMP
    export PYTHONPATH=/home/firedrake/firedrake/lib/python3.6/site-packages
%post
    apt-get update
    apt-get install -y git build-essential wget libdapl2 dapl2-utils libdapl-dev linux-headers-generic libnuma-dev mesa-utils freeglut3-dev libvtk7-dev python3-pip
    ln -s /usr/bin/pip3 /usr/bin/pip
    export CC=gcc
    export CXX=g++

    # Build & install libpsm2 for omnipath support
    git clone https://github.com/intel/opa-psm2.git \
    && cd opa-psm2 \
    && make -j4 && make install
    if [ ! -d /opa-psm2 ]; then
        echo "Failed to clone PSM2"
        exit 1
    fi

    # Install extra software for firedrake
    pip3 install scipy
    pip3 install vtk
    mkdir /tmp/firedrake_testcache
    ln -s /tmp/firedrake_testcache /home/firedrake/firedrake/.cache

    # hack firedrake activation into our container for both login and interactive non-login shells
    echo "source /home/firedrake/firedrake/bin/activate" >> /etc/bash.bashrc
    echo "source /home/firedrake/firedrake/bin/activate" >> /etc/profile

    # clean up
    rm -rf /var/lib/apt/lists/*
    rm -rf /opa-psm2

%labels
    Author Sven Hansen
    Version v0.13
```

Findings

- Existing Docker infrastructure makes adoption easy
  But: Special software still needs special care

- User profiles and configs can be false friends
  - Strict usage of %environment section helps
  - Using zsh as the standard shell on Claix circumvented some problems

- Optimal performance may require additional preparation, e.g. OmniPath support
  - Adding libraries can be streamlined to enhance containers

- Containers are low maintenance as long as the containerized software is

Thank You
for your attention