Docker and Docker-compose

Purpose

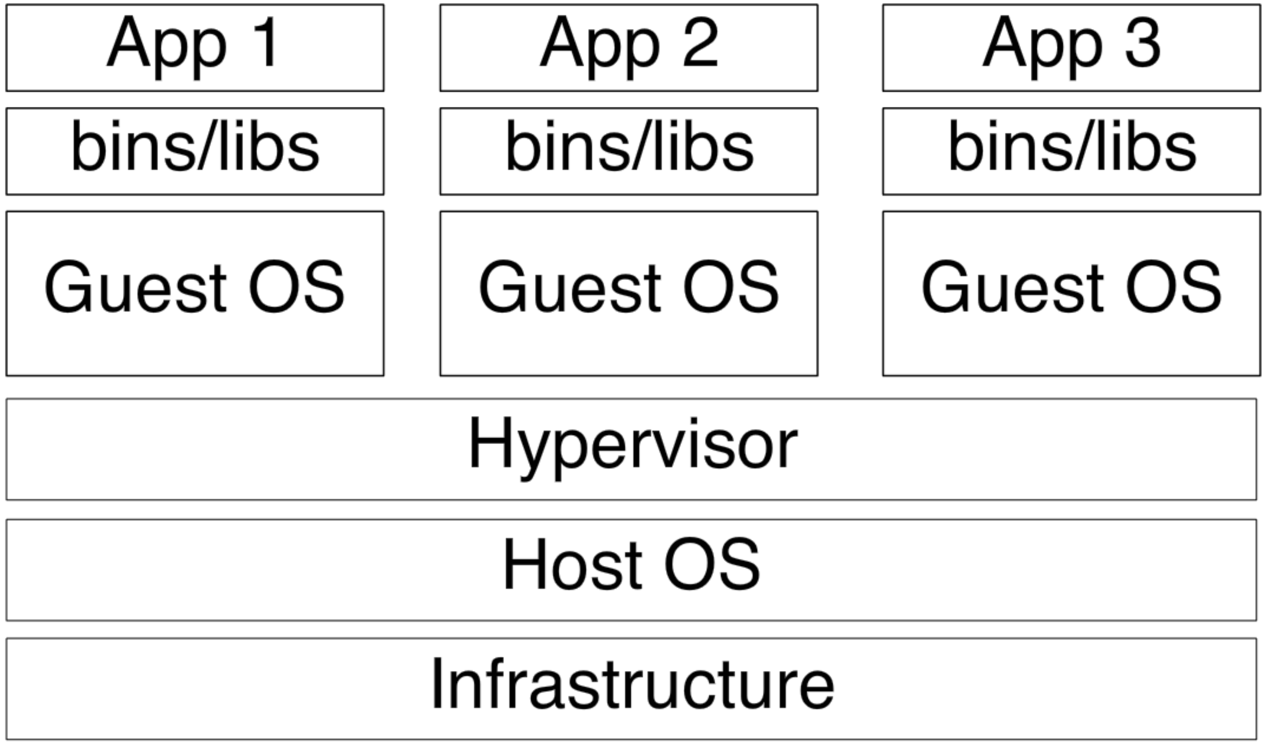
- Deploy, test, and ship your software faster
- Compared with previous students of this course, you shall save hundreds of hours in this project by using Docker and Docker-compose!

What is Docker?

- Docker is an open-source project that automates the deployment of applications inside software containers
- Docker is a tool for running applications in an isolated environment (Docker containers) on the host operating system
- From Docker official page: *Docker containers wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries – anything that can be installed on a server. This guarantees that the software will always run the same, regardless of its environment.*

Virtual machine (VM)

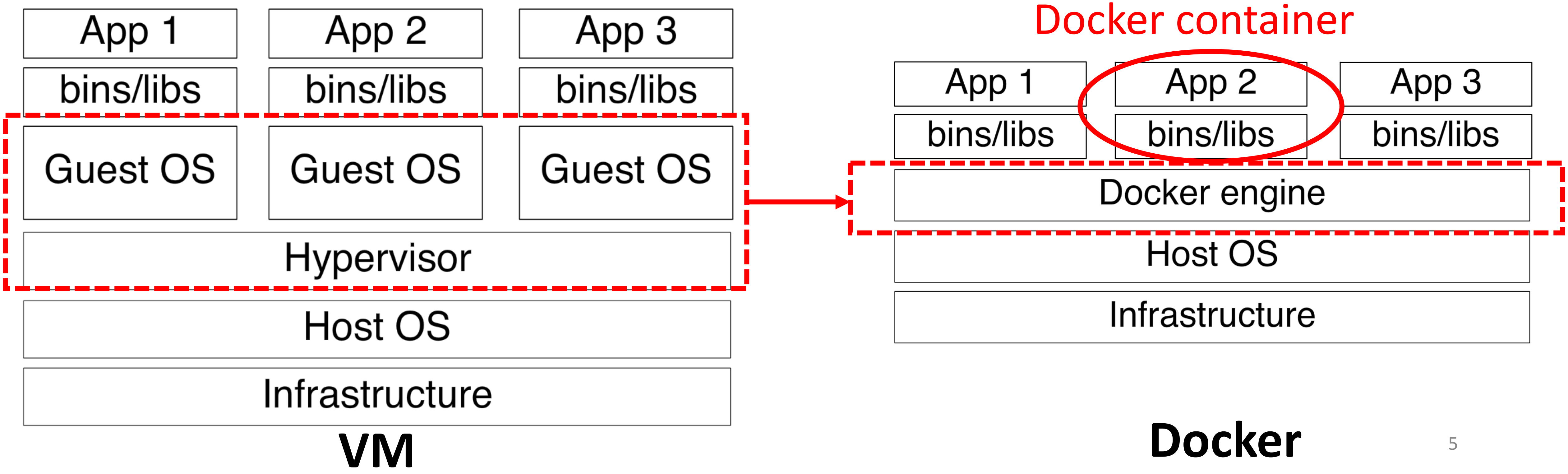
- Isolation between applications
- E.g., VirtualBox
- Separate OS kernels
- Huge disk consumption
- It takes minutes to start a VM



VM

Virtual machine vs Docker

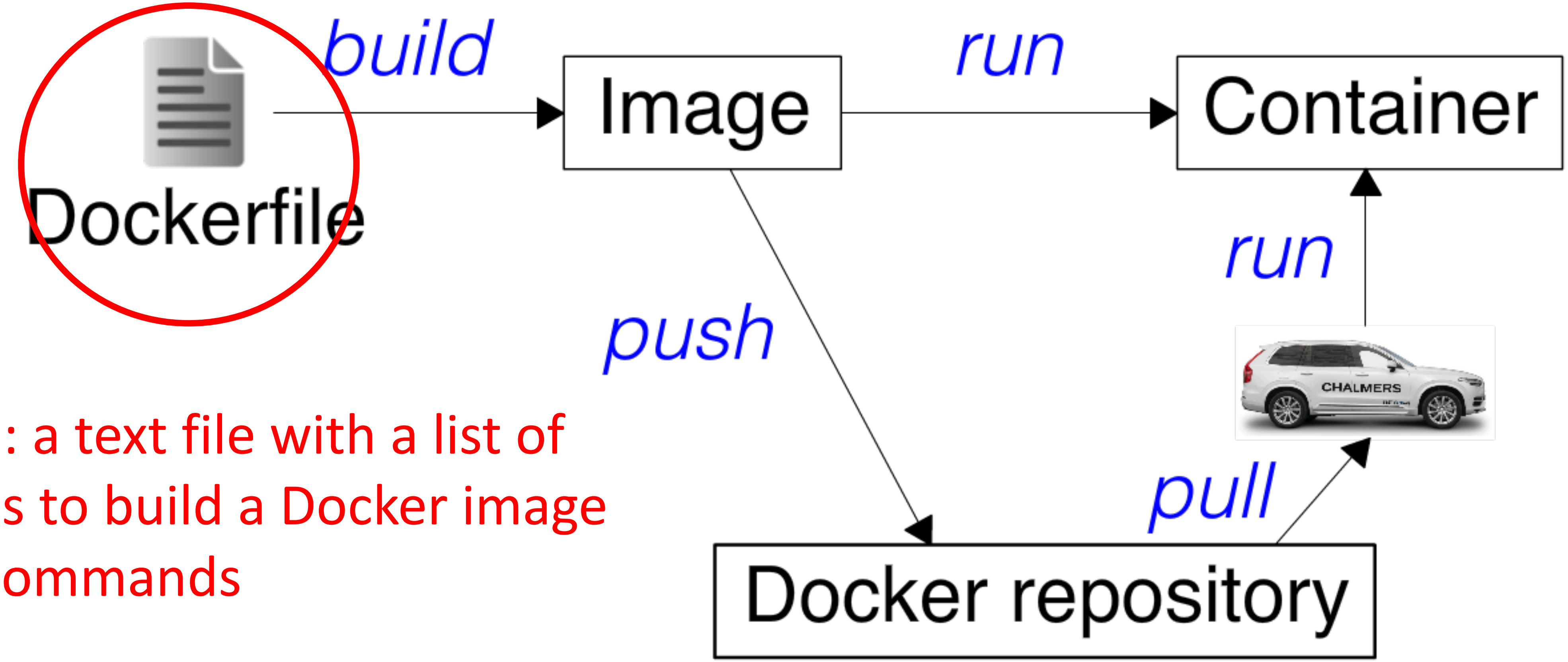
- Docker also provides resource isolation (isolated processes)
- A Docker container includes the application and all of its dependencies
- Shared kernel between containers and the host machine
- Start in seconds; much less CPU and disk consumption



Why Docker?

- Your software always runs in the same environment
  - If it works on one computer, it works on other computers as well
- No need to install all the dependencies
  - Skip the setup of the software development environment
- Easy to rollback to a previous working version when problems occur
- Secure due to isolation between Docker containers; almost no effect on the host machine

Docker images and containers



Dockerfile

build

Image

run

Container

push

run

pull

Docker repository

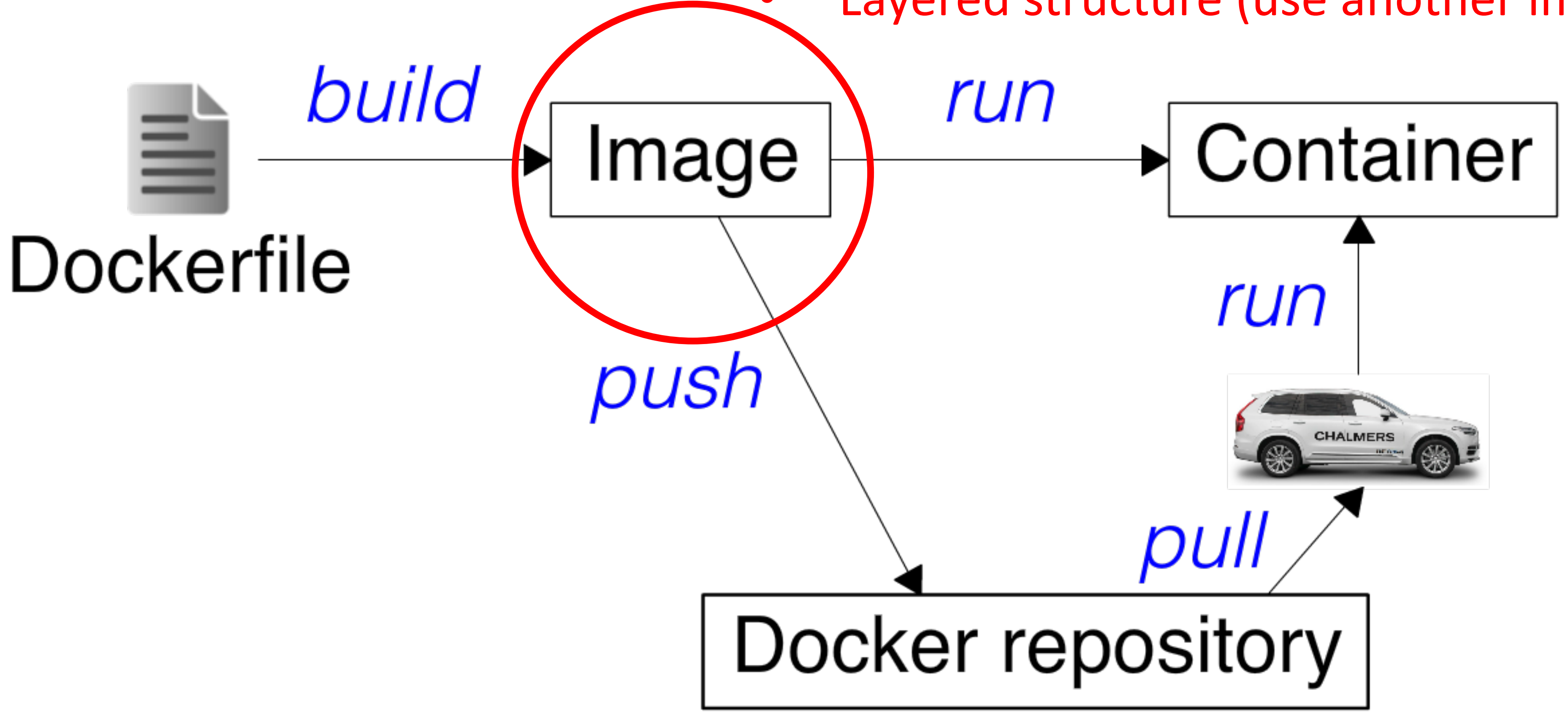
Dockerfile: a text file with a list of commands to build a Docker image

- Linux commands

Docker images and containers

Docker image:

- A static file system built from a Dockerfile
- Can be built, pushed, and pulled
- With tags/versions (e.g., latest, v0.1.0)
- Layered structure (use another image as the base image)



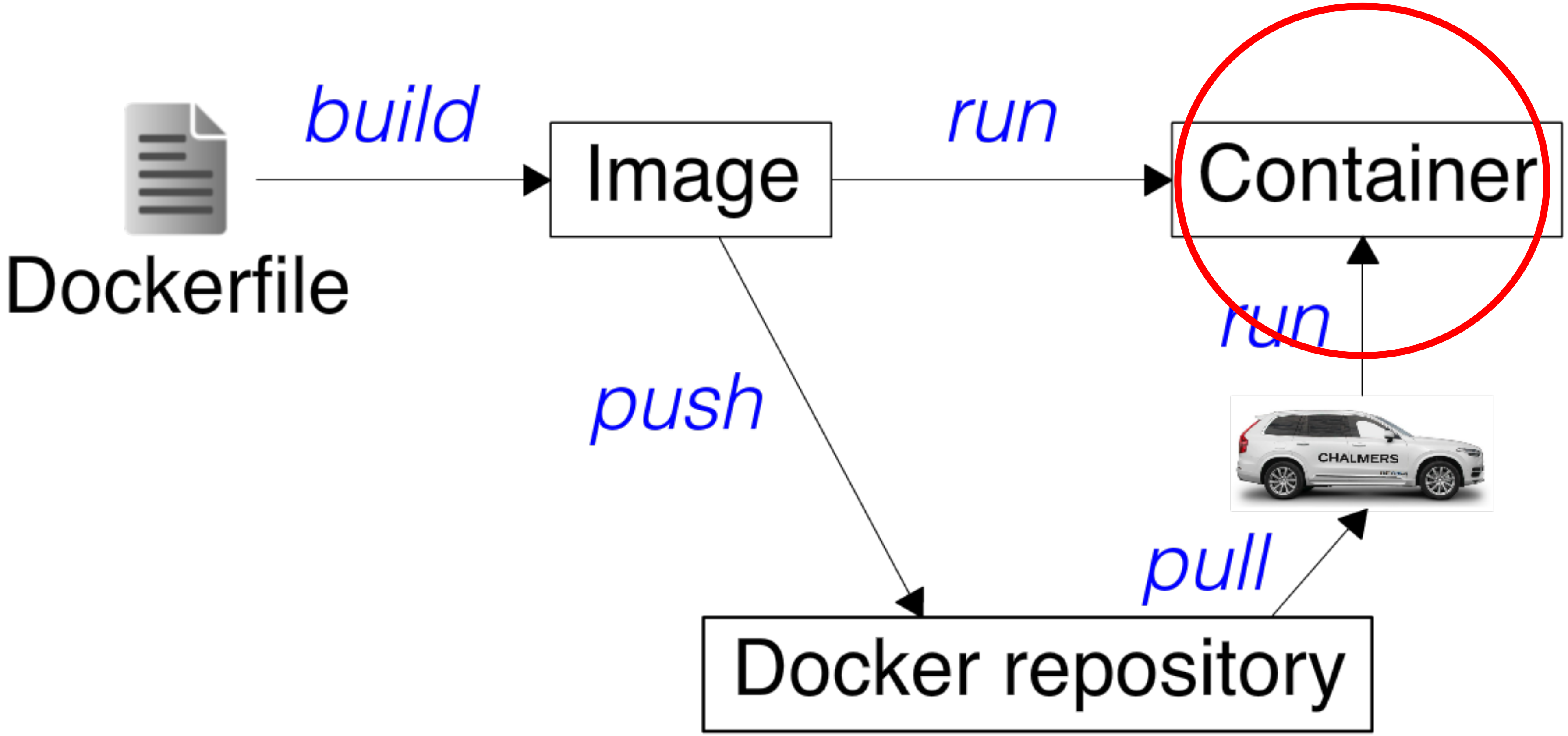
Dockerfile of seresearch/opendavinci-ubuntu-16.04

- seresearch/opendavinci-ubuntu-16.04: Ubuntu 16.04 + packages required to build OpenDaVINCI
- GitHub page: https://github.com/opendavinci/opendavinci-ubuntu-16.04
- OpenDaVINCI uses seresearch/opendavinci-ubuntu-16.04 as the base image: https://github.com/se-research/OpenDaVINCI/blob/master/docker/Makefile
  - BASE_IMAGE=seresearch/opendavinci-ubuntu-16.04
  - BASE_IMAGE_VERSION=latest

Docker images and containers

Docker container: a running instance of the Docker image

- Created when you run a Docker image



Common Docker commands

- By default you need sudo to run Docker
- Do not forget to add your user account to the Docker group to run Docker without sudo: https://docs.docker.com/engine/installation/linux/linux-postinstall/

- docker push/pull: push/pull an image to/from a repository
- docker images: see a list of all images on your system
- docker rmi <image ID>: remove a Docker image
  - Use -f if necessary
  - It is sufficient to indicate only the first 3 digits of an image ID
- docker run: run a Docker container (docker run --help: see list of all flags)
  - It will try to fetch the image from the repository if the specified image does not exist locally
  - -d: run a container in detached mode
- docker stop <Container ID>: stop a detached container
- docker rm <Container ID>: delete a container

Rule of thumb: Clean up containers once you are done with them--stopped and removed

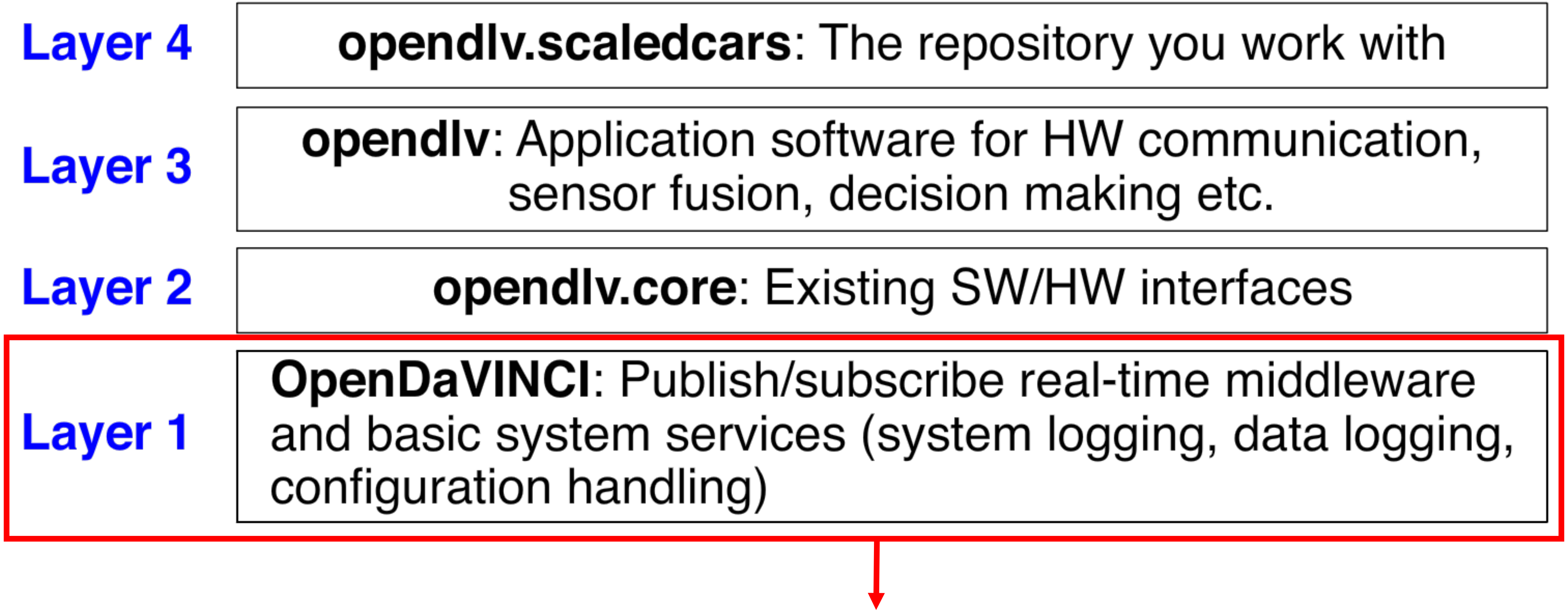
Common Docker commands

- Use the *--rm* flag in *docker run* to automatically delete the container when it is stopped
- docker ps: show all the containers that are currently running
- docker ps -a: show all the containers that we ran

Combine commands to stop all containers, and delete all containers/images:

- Stop all containers: docker stop $(docker ps -a -q) or docker stop $(docker ps -aq)
- Delete all containers: docker rm $(docker ps -a -q)
- Delete all images: docker rmi $(docker images -a -q) →Think carefully before you do this

Software layers



Layer 4 — **opendlv.scaledcars**: The repository you work with

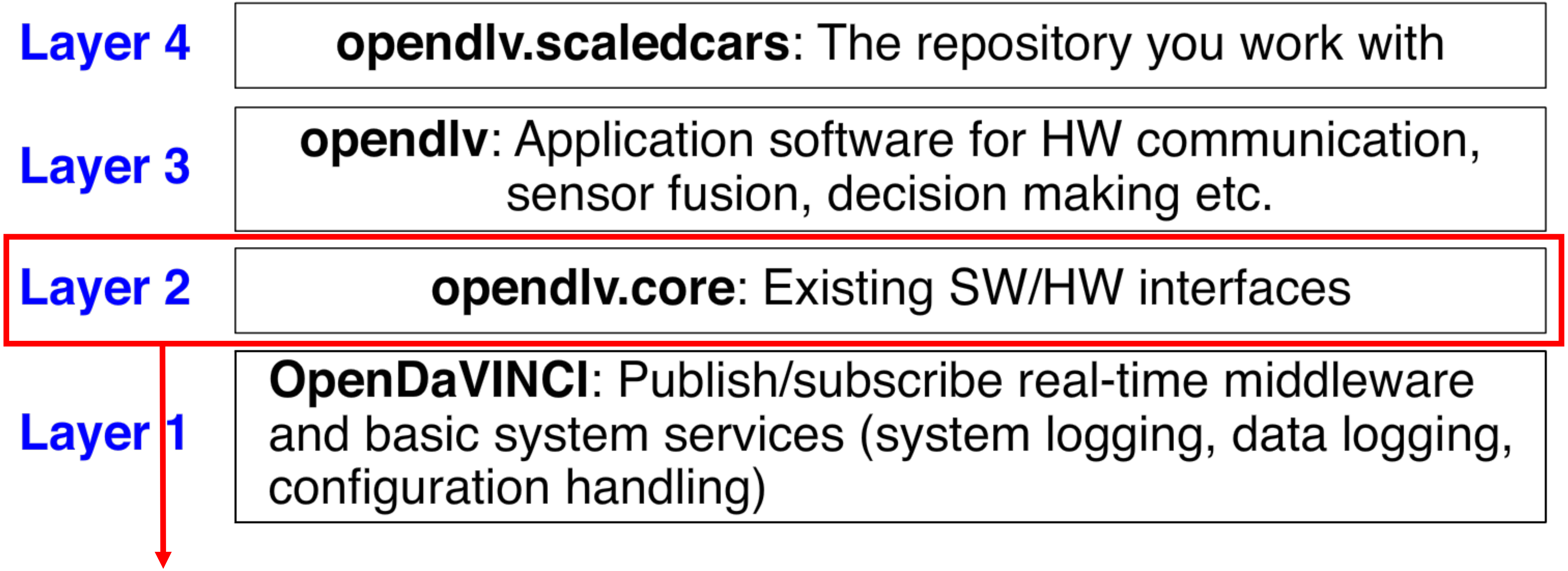
Layer 3 — **opendlv**: Application software for HW communication, sensor fusion, decision making etc.

Layer 2 — **opendlv.core**: Existing SW/HW interfaces

Layer 1 — **OpenDaVINCI**: Publish/subscribe real-time middleware and basic system services (system logging, data logging, configuration handling)

GitHub: https://github.com/se-research/OpenDaVINCI
Docker repository: https://hub.docker.com/r/seresearch/opendavinci-ubuntu-16.04-complete/

Software layers

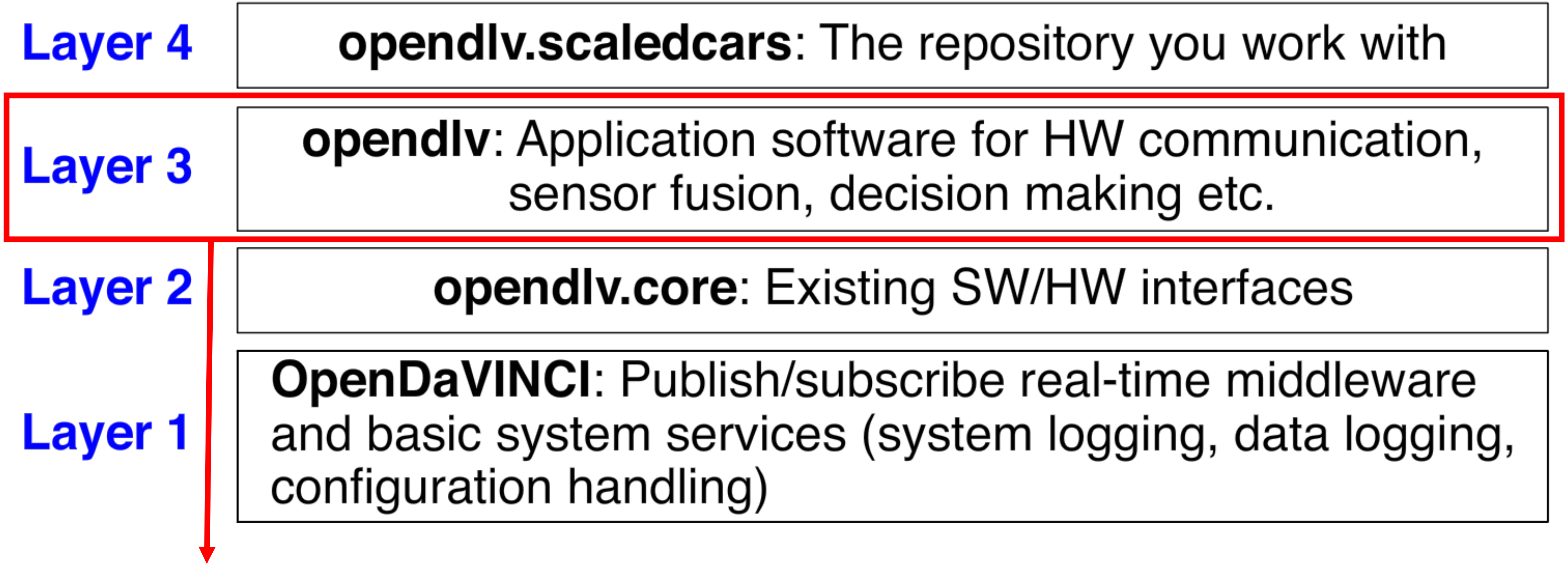


Layer 4	**opendlv.scaledcars**: The repository you work with
Layer 3	**opendlv**: Application software for HW communication, sensor fusion, decision making etc.
Layer 2	**opendlv.core**: Existing SW/HW interfaces
Layer 1	**OpenDaVINCI**: Publish/subscribe real-time middleware and basic system services (system logging, data logging, configuration handling)

GitHub: https://github.com/chalmers-revere/opendlv.core

Docker repository: https://hub.docker.com/r/seresearch/opendlv-core-on-opendavinci-ubuntu-16.04-complete/

Software layers



Layer 4 — **opendlv.scaledcars**: The repository you work with

Layer 3 — **opendlv**: Application software for HW communication, sensor fusion, decision making etc.

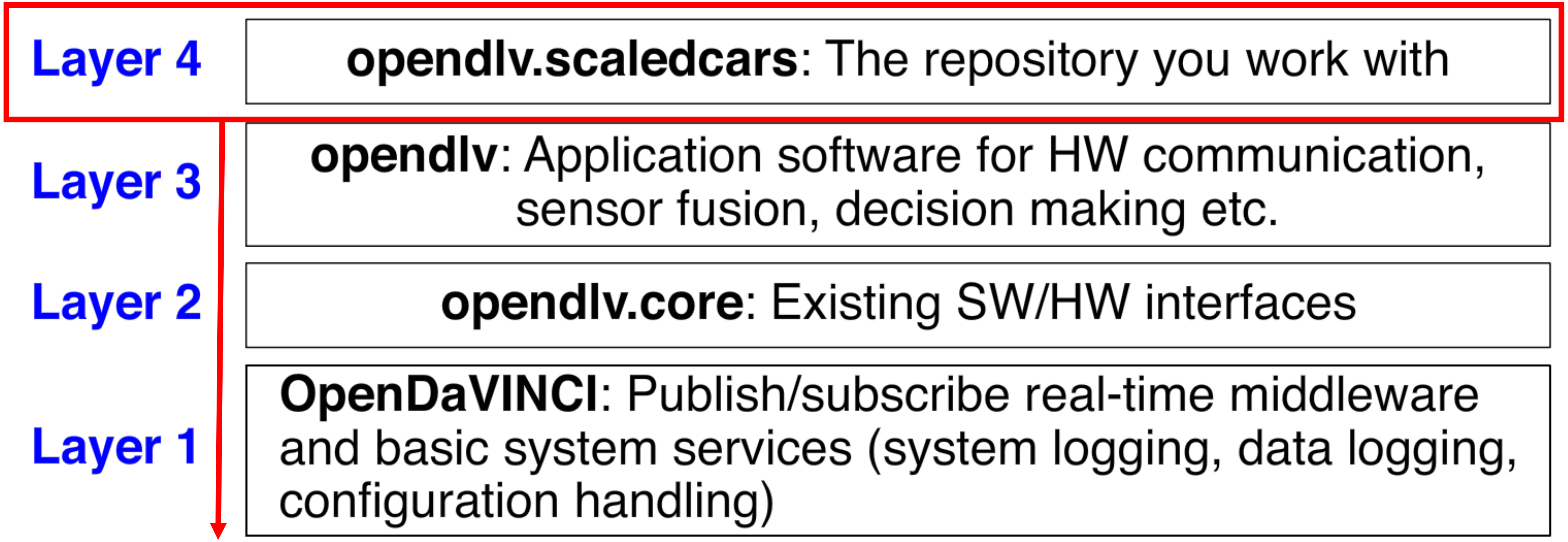
Layer 2 — **opendlv.core**: Existing SW/HW interfaces

Layer 1 — **OpenDaVINCI**: Publish/subscribe real-time middleware and basic system services (system logging, data logging, configuration handling)

GitHub: https://github.com/chalmers-revere/opendlv

Docker repository: https://hub.docker.com/r/seresearch/opendlv-on-opendlv-core-on-opendavinci-ubuntu-16.04-complete/

Software layers



Layer 4 — **opendlv.scaledcars**: The repository you work with

Layer 3 — **opendlv**: Application software for HW communication, sensor fusion, decision making etc.

Layer 2 — **opendlv.core**: Existing SW/HW interfaces

Layer 1 — **OpenDaVINCI**: Publish/subscribe real-time middleware and basic system services (system logging, data logging, configuration handling)

GitHub: https://github.com/chalmers-revere/opendlv.scaledcars

- Fork from master for your own group

Docker repository: https://hub.docker.com/r/seresearch/scaledcars-on-opendlv-core-on-opendavinci-ubuntu-16.04-complete/

- Create a separate Docker repository for your own group

Be aware of new releases of opendlv

- opendlv is used as the base image for opendlv.scaledcars
- opendlv releases new versions from time to time
- Update opendlv.scaledcars/docker/Makefile in the source tree:
  - BASE_IMAGE=seresearch/opendlv-on-opendlv-core-on-opendavinci-ubuntu-16.04-complete
  - BASE_IMAGE_VERSION=xxx

Source tree structure of opendlv.scaledcars

- cmake.Modules: compiler setup (e.g., C++ compiler flags); will be explained more in later lectures
- code: your application code for lane following, parking, overtaking…
- docker: build your code and create the Docker image
- resources: define data structures used in your application
- usecases: will be explained when I talk about Docker-compose soon

How to locally build your software and create the Docker image?

1. Go to the docker folder in the source tree
2. Run make buildComplete to build your software in a Docker container
   - The binaries built from your code can be found at: opendlv.scaledcars/docker/builds/scaledcars-on-opendlv-on-opendlv-core-on-opendavinci-ubuntu-16.04-complete-master/opt/opendlv.scaledcars/bin
3. If you make some changes after make buildComplete, run make buildIncremental instead to speed up the building process
4. Run make createDockerImage to create the Docker image locally
5. Commands can be combined: make buildComplete createDockerImage
6. Specify a specific version of the base image: make BASE_IMAGE_VERSION=xxx buildComplete createDockerImage
7. Run make cleanAll to clean the builds folder and Docker images

Manage your GitHub repository

- Each group forks https://github.com/chalmers-revere/opendlv.scaledcars
- Set up your repository in a way that no one is allowed to directly change the master branch
- Individual group members create branches from the master branch for own experiments, e.g., 2017.Q1.feature.laneFollowing
- To integrate a change in a branch, make a Pull Request (PR) from the branch to master. Assign at least another group member as the reviewer of the PR. **Review the PR carefully!**
- The change is present in master after the PR is approved and the branch is merged into master

Manage your GitHub repository: example

1. git checkout -b 2017.Q1.feature.laneFollowing (create a new branch)
2. git push origin 2017.Q1.feature.laneFollowing (other people can also change this branch now)
3. Make changes in 2017.Q1.feature.laneFollowing
4. git add <Files_Changed> (can be repeated if multiple files are changed)
5. git commit -m “A commit message”
6. git pull origin 2017.Q1.feature.laneFollowing (in case other people have made changes)
7. git push origin 2017.Q1.feature.laneFollowing
8. git checkout master
9. git pull (get the latest update of the master branch)
10. git checkout -b 2017.Q1.feature.laneFollowing
11. git merge master + rebuild the repository locally
12. Create a PR to be merged into master after a successful build in Step 11

Manage your GitHub repository: tips

- Use git checkout to revert a change
- Use git status to check which files are changed
- Use git diff to check what has been changed in a file
- Use git pull before git push to minimize the risk of conflicts
- Never use -f!
- Follow some Git tutorials, practice, and learn from each other

Demo: record and replay video data with an OpenCV camera

Software components for the video recording and replay

SW component	Description	Layer	Record/Replay
odsupercomponent	Lifecycle management; establish UDP multicast	OpenDaVINCI	both
proxy-camera	Interface with the camera	opendlv.core	record
odrecorderh264	Store video data in H264 format	OpenDaVINCI	record
odcockpit(+odplayerh264)	Replay the recorded video	OpenDaVINCI	replay

Approach 1: without Docker

- Step 1: Install all the required packages for building OpenDaVINCI
  - ant build-essential cmake default-jre default-jdk freeglut3 freeglut3-dev git libboost-dev libopencv-dev libopencv-core-dev libopencv-highgui-dev libopencv-imgproc-dev libpopt-dev libqt4-dev libqt4-opengl-dev libqwt5-qt4-dev libqwt5-qt4 qt4-dev-tools rpm psmisc wget ffmpeg
  - Instruction: http://opendavinci.readthedocs.io/en/latest/installation.html
- Step 2: Build OpenDaVINCI
  1. Go to /opt, and create an installation folder sudo mkdir od
  2. Give write permission to this folder sudo chown $USER:$USER /opt/od
  3. Go to OpenDaVINCI source folder and create a build folder: mkdir build && cd build
  4. Use cmake to create the build scripts: cmake -D CMAKE_INSTALL_PREFIX=/opt/od ..
  5. make (ca 40min)

Approach 1: without Docker

- Step 3: Build opendlv.core to get the proxy-camera binary
  - Go to the opendlv.core/docker folder in the opendlv.core source tree
  - Build opendlv.core within a Docker container: make buildComplete
  - make buildIncremental (the binary proxy-camera can be found at: opendlv.core/docker/builds/opendlv-core-on-opendavinci-ubuntu-16.04-complete-master/opendlv.core.build/build.system/proxy-camera)
- Step 4: Specify parameters for all SW components in /opt/od/bin/configuration

Configuration parameters for proxy-camera

```
proxy-camera.camera.debug = 0      # 1 = show recording (requires X11), 0 = otherwise.
proxy-camera.camera.name = DocumentationCamera0
proxy-camera.camera.id = 0         # Select here the proper ID for OpenCV.
proxy-camera.camera.width = 640
proxy-camera.camera.height = 480
proxy-camera.camera.bpp = 3
proxy-camera.camera.flipped = 1    # 1 = flipped image, 0 = not flipped image.
```

Approach 1: without Docker

- Step 5: Video recording
  - Terminal 1: LD_LIBRARY_PATH=/opt/od/lib /opt/od/bin/odsupercomponent --cid=111 --verbose=1 --configuration=/opt/od/bin/configuration
  - Go to the proxy-camera binary folder and in Terminal 2 run: LD_LIBRARY_PATH=/opt/od/lib ./opendlv-core-system-proxy-camera --cid=111 --freq=20
  - Terminal 3 at /opt/od/bin: LD_LIBRARY_PATH=/opt/od/lib ./odrecorderh264 --cid=111
  - Ctrl + C in Terminal 1 to stop the recording (a .rec file, an empty .rec.mem file, and a .h264 file at /opt/od/bin)
- Step 6: Video replay
  - Terminal 1: LD_LIBRARY_PATH=/opt/od/lib /opt/od/bin/odsupercomponent --cid=111 --verbose=1 --configuration=/opt/od/bin/configuration
  - Terminal 2: LD_LIBRARY_PATH=/opt/od/lib /opt/od/bin/odcockpit --cid=111
  - Open SharedImageViewer and Player plugins. Load the .rec file and view the video in SharedmageViewer

Approach 2: with Docker

Assume a clean Ubuntu 16.04 with Docker installed

- Step 1: docker pull seresearch/opendlv-core-on-opendavinci-ubuntu-16.04-complete
- Step 2 (Terminal 1): docker run -ti --rm --net=host -v /opt/od/bin:/opt/opendlv.core.configuration seresearch/opendlv-core-on-opendavinci-ubuntu-16.04-complete:latest /opt/od4/bin/odsupercomponent --cid=111 --verbose=1 --configuration=/opt/opendlv.core.configuration/configuration
- Step 3 (Terminal 2): docker run -ti --rm --net=host --ipc=host --user=odv --group-add video --device=/dev/video0:/dev/video0 seresearch/opendlv-core-on-opendavinci-ubuntu-16.04-complete:latest /opt/opendlv.core/bin/opendlv-core-system-proxy-camera --cid=111 --freq=20
- Step 4 (Terminal 3): docker run -ti --rm --net=host --ipc=host --user=odv -v ~/recordings:/opt/recordings -w /opt/recordings seresearch/opendlv-core-on-opendavinci-ubuntu-16.04-complete:latest /opt/od4/bin/odrecorderh264 --cid=111
- Step 5: Ctrl + C in Terminal 1 to stop the recording (recording files at ~/recordings)

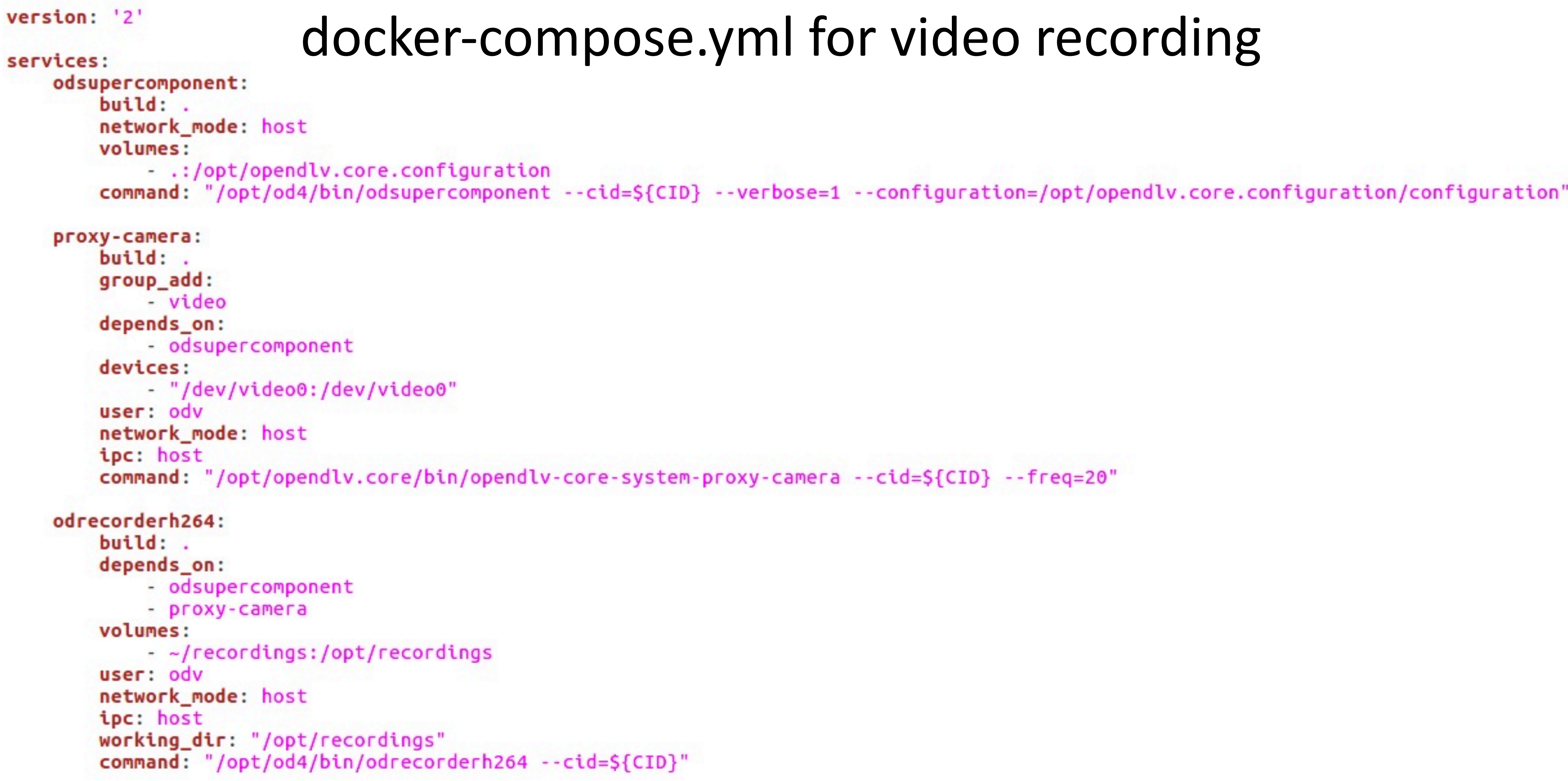
Approach 2: with Docker

- Step 6: Video replay
  - Terminal 1: docker run -ti --rm --net=host -v /opt/od/bin:/opt/opendlv.core.configuration seresearch/opendlv-core-on-opendavinci-ubuntu-16.04-complete:latest /opt/od4/bin/odsupercomponent --cid=111 --verbose=1 --configuration=/opt/opendlv.core.configuration/configuration
  - Terminal 2: run xhost + to grant Docker access to the Xserver
  - docker run -ti --rm --net=host --ipc=host --user=odv -e DISPLAY=$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix -v ~/recordings:/opt/recordings seresearch/opendlv-core-on-opendavinci-ubuntu-16.04-complete:latest /opt/od4/bin/odcockpit --cid=111
  - Open SharedImageViewer and Player plugins. Load the .rec file (/opt/recordings) and view the video in SharedmageViewer

Docker-compose

- A tool for defining and running multi-container Docker applications
- Run an “App” (multiple containers) instead of a single container
- Need to be installed separately in addition to Docker
- Start different Docker containers as micro-services in the same network
- Specify the base Docker image in a Dockerfile
  - FROM seresearch/opendlv-core-on-opendavinci-ubuntu-16.04-complete:v0.8.5
- Specify micro-services in a docker-compose.yml file

docker-compose.yml for video recording



```
version: '2'

services:
    odsupercomponent:
        build: .
        network_mode: host
        volumes:
            - .:/opt/opendlv.core.configuration
        command: "/opt/od4/bin/odsupercomponent --cid=${CID} --verbose=1 --configuration=/opt/opendlv.core.configuration/configuration"

    proxy-camera:
        build: .
        group_add:
            - video
        depends_on:
            - odsupercomponent
        devices:
            - "/dev/video0:/dev/video0"
        user: odv
        network_mode: host
        ipc: host
        command: "/opt/opendlv.core/bin/opendlv-core-system-proxy-camera --cid=${CID} --freq=20"

    odrecorderh264:
        build: .
        depends_on:
            - odsupercomponent
            - proxy-camera
        volumes:
            - ~/recordings:/opt/recordings
        user: odv
        network_mode: host
        ipc: host
        working_dir: "/opt/recordings"
        command: "/opt/od4/bin/odrecorderh264 --cid=${CID}"
```

Use cases for Docker-compose

- Define CID in a .env file (e.g. CID=111)
- Prepare a use case folder including docker-compose.yml, Dockerfile, and all necessary files (e.g., the configuration file)
- Use case for video recording: opendlv.core/usecases/stable/documentationcamera/recording.documentationcamera/
- Start recording: docker-compose up --build
- Stop recording: (1) ctrl + C; (2) docker-compose stop; (3) docker-compose rm
- Combine (2) and (3): docker-compose stop; yes|docker-compose rm

Video replay using Docker-compose

- Use case: opendlv.core/usecases/stable/replay.recording
- Start replaying: docker-compose up --build
- In Player of odcockpit, load the recording from /opt/recordings
- Stop replaying: (1) ctrl + C; (2) docker-compose stop; (3) docker-compose rm

Use cases for video recording and replay in opendlv.scaledcars instead of opendlv.core

- At opendlv.scaledcars/usecases:
  - Video recording: recording.documentationcamera
  - Video replay: replay.recording
- Both use cases use Layer 4 Docker image: seresearch/scaledcars-on-opendlv-on-opendlv-core-on-opendavinci-ubuntu-16.04-complete:latest, which includes Layer 2 Docker image seresearch/opendlv-core-on-opendavinci-ubuntu-16.04-complete used in the Demo

Instructions for building opendlv.scaledcars and running both use cases

1. git clone https://github.com/chalmers-revere/opendlv.scaledcars.git
2. cd opendlv.scaledcars/docker
3. make buildComplete
4. make createDockerImage
5. Check Docker images: docker images
6. cd ../usecases/recording.documentationcamera
7. Record video: docker-compose up --build
8. Stop recording: docker-compose stop and docker-compose rm
9. Check recordings in ~/recordings
10. Go to usecases/replay.recording
11. Start replay: docker-compose up --build
12. In player of odcockpit, load the recording from /opt/recordings
13. Stop replay: docker-compose stop and docker-compose rm

Summary

- Docker (what + why)
- Docker vs VM (Virtual Machine)
- Dockerfile, Docker image, Docker container, Docker repository
- Docker commands
- Software layers
- opendlv.scaledcars: source tree structure
- Build software and create Docker images
- GitHub workflow and management
- Docker-compose
- Demo: video recording and replay
  - Approach 1: non-Docker
  - Approach 2: Docker run
  - Approach 3: Docker-compose

Thank you!