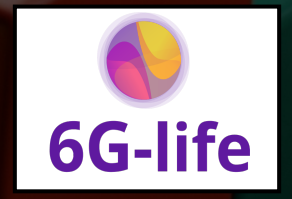
On-Demand Container Partitioning for Distributed ML

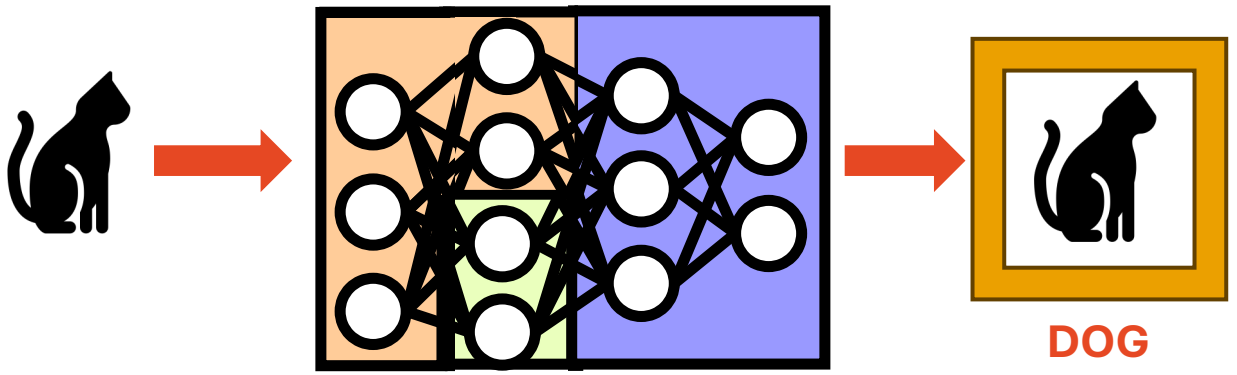
Giovanni Bartolomeo[#*]
Navidreza Asadi[#*]
Wolfgang Kellerer[#]
Jörg Ott[#]
Nitinder Mohan[Δ]

Technical University of Munich

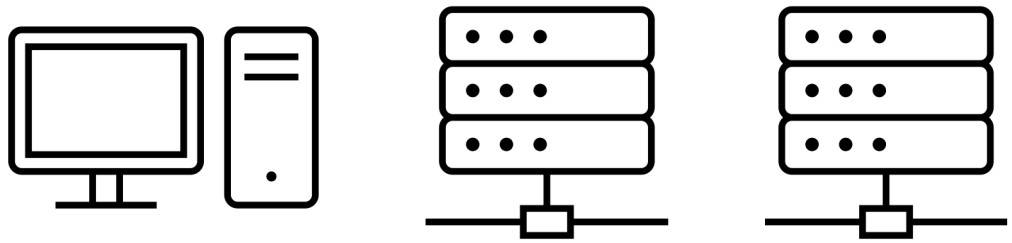


* Equal Contribution
#Technical University of Munich, Germany, {name.surname}@tum.de
ΔTU Delft, The Netherlands, n.mohan@tudelft.nl

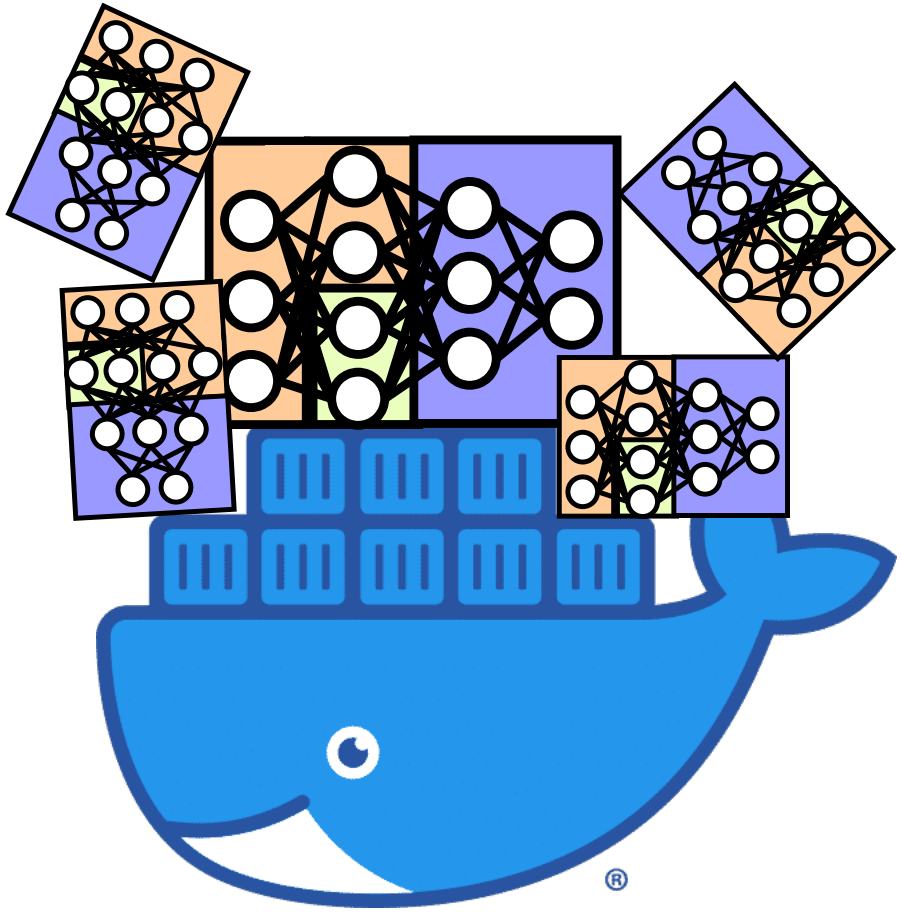
Split NN



- ML models can be partitioned
- Model partitions can be distributed independently
- Partitioned models can better fit real-time hardware availability
- Improved resource utilization in shared infrastructures
- Improved inference scalability
- Practical deployment and management of these model partitions is still challenging and a fairly manual task



Why Containers?



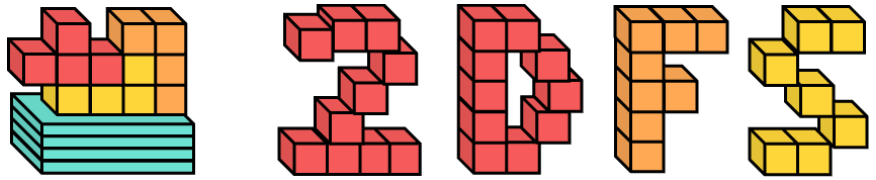
- Strong momentum towards OCI (open container initiative) artifacts for model weights packaging [1]
- Familiar environment for develoepers
- Unified lifecycle between application and its model
- Pre-existing distribution infrastructure via container registries
- Applications will mostly use containers anyway
- MLOps, Cloud storage solutions and Volumes are still an option, but there are some tradeoffs [more details in our paper]

[1] Emily Casey, Docker - **Why Docker Chose OCI Artifacts for AI Model Packaging**

Why (NOT) Containers?

- Frequent model updates will trigger image re-builds
  - Model updates can happen extremely frequently in the real world (e.g. every minute)
- Model updates can happen at different times compared to code updates
- Split computing requires us to build one image for each model partition for each possible partition combination
- ML model weigths are not the only problem, what about static binaries and drivers that we package together? Our containers are bloated! [2]

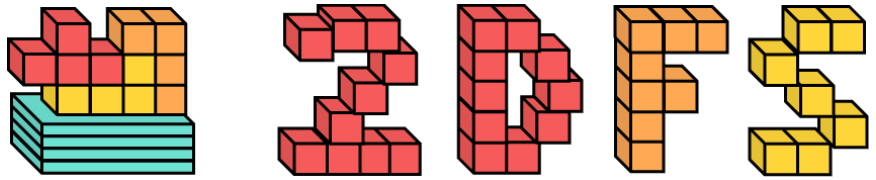
[2] Huaifeng Zhang et al. **Machine learning systems are bloated and vulnerable**



“

2DFS is a two-dimensional filesystem build and distribution framework for containers

”



”

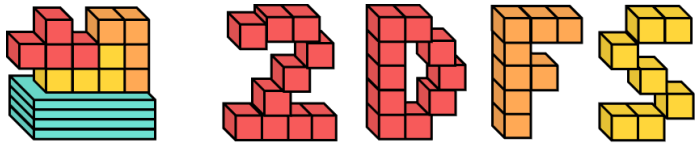
2DFS is a two-dimensional filesystem build and distribution framework for containers

”

New OCI Layer:
`2dfs.field`

2DFS Builder
Build OCI+2DFS images

2DFS Registry
Offers live images partitioning



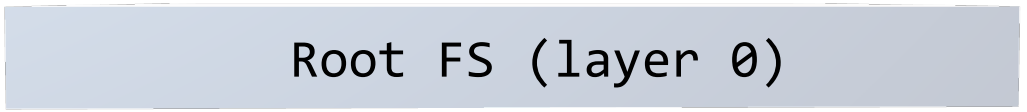
According to the OCI specification

A container image is composed of:

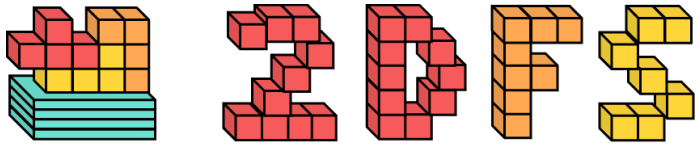
- Image manifest
- Image index
- Configuration
- Filesystem layers

The Filesystem layers are composed of:

- A root FS base layer (parent)
- Zero or more layers are applied on top representing the changesets from the base Root FS



OCI Filesystem layers



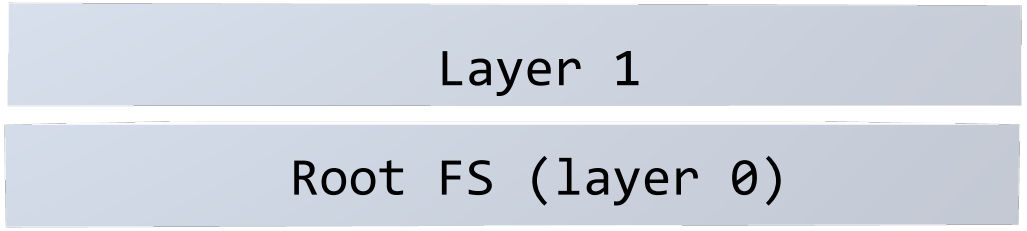
According to the OCI specification

A container image is composed of:

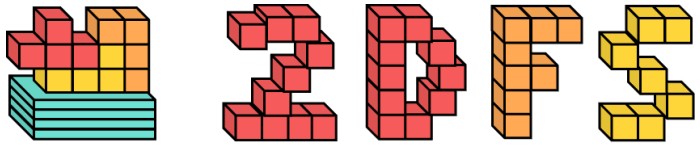
- Image manifest
- Image index
- Configuration
- Filesystem layers

The Filesystem layers are composed of:

- A root FS base layer (parent)
- Zero or more layers are applied on top representing the changesets from the base Root FS



OCI Filesystem layers

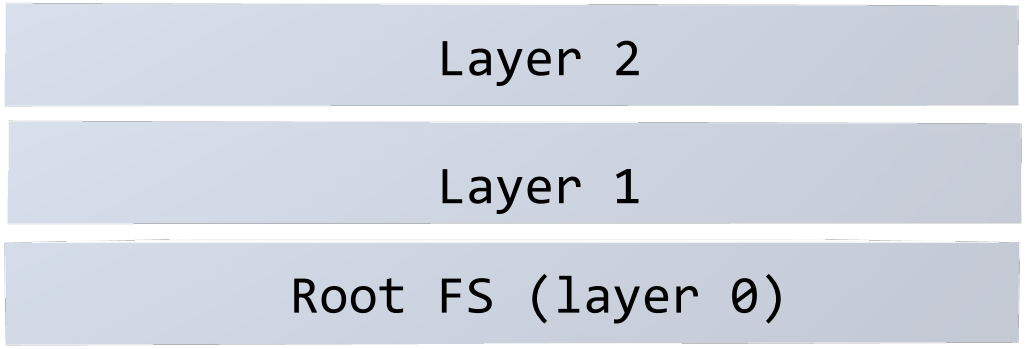


According to the OCI specification
A container image is composed of:

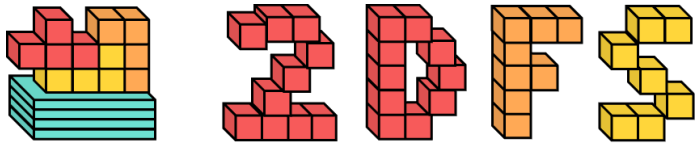
- Image manifest
- Image index
- Configuration
- Filesystem layers

The Filesystem layers are composed of:

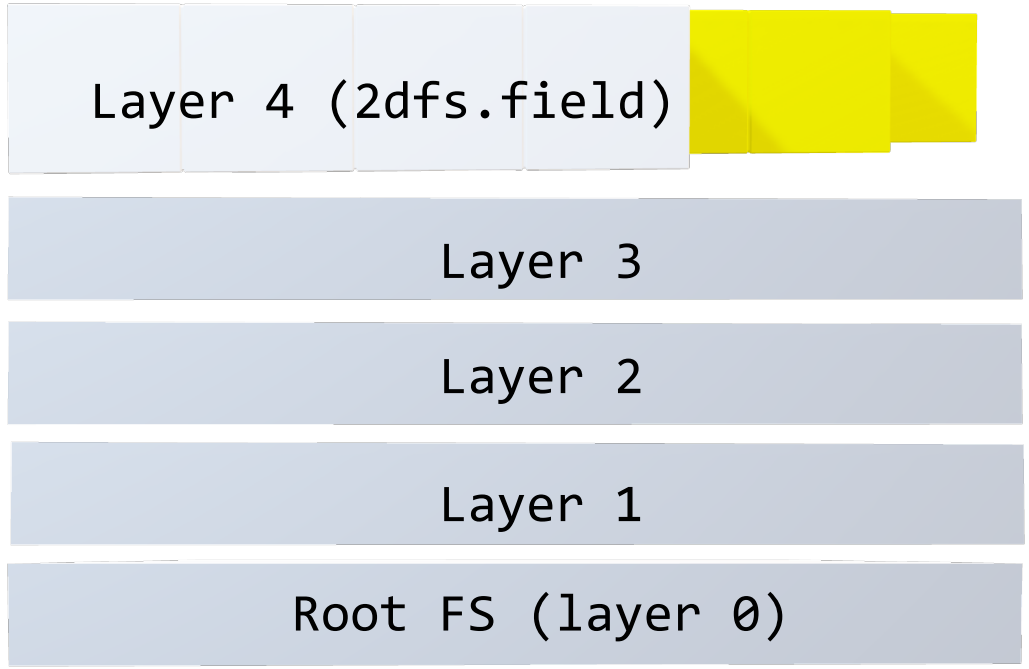
- A root FS base layer (parent)
- Zero or more layers are applied on top representing the changesets from the base Root FS
- If something changes on layer 1, the cache for all the layers above is invalidated



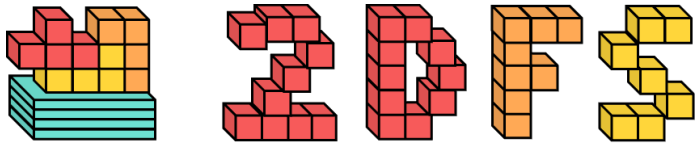
OCI Filesystem layers



2dfs.field
Is just a new layer type

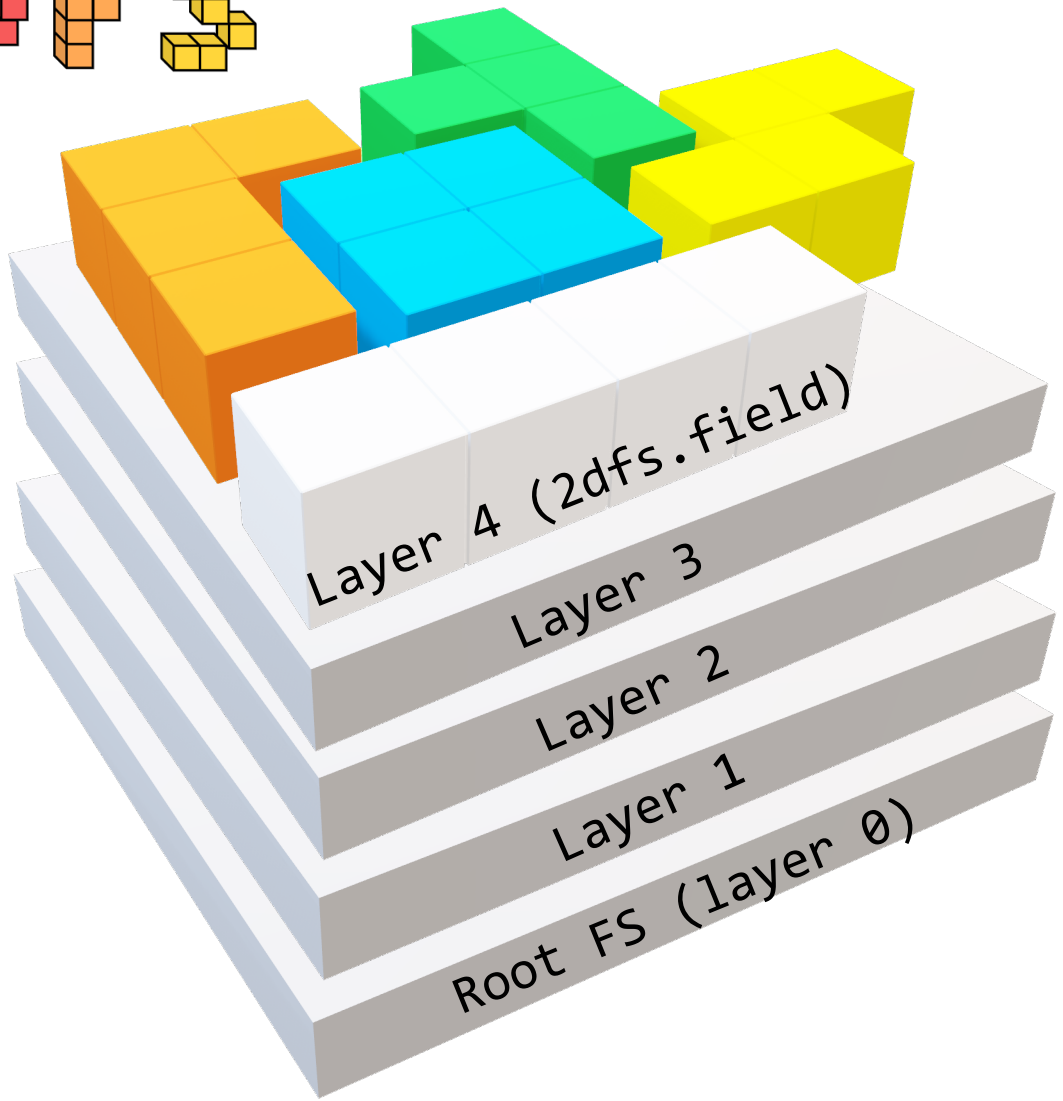


OCI+2DFS Filesystem layers



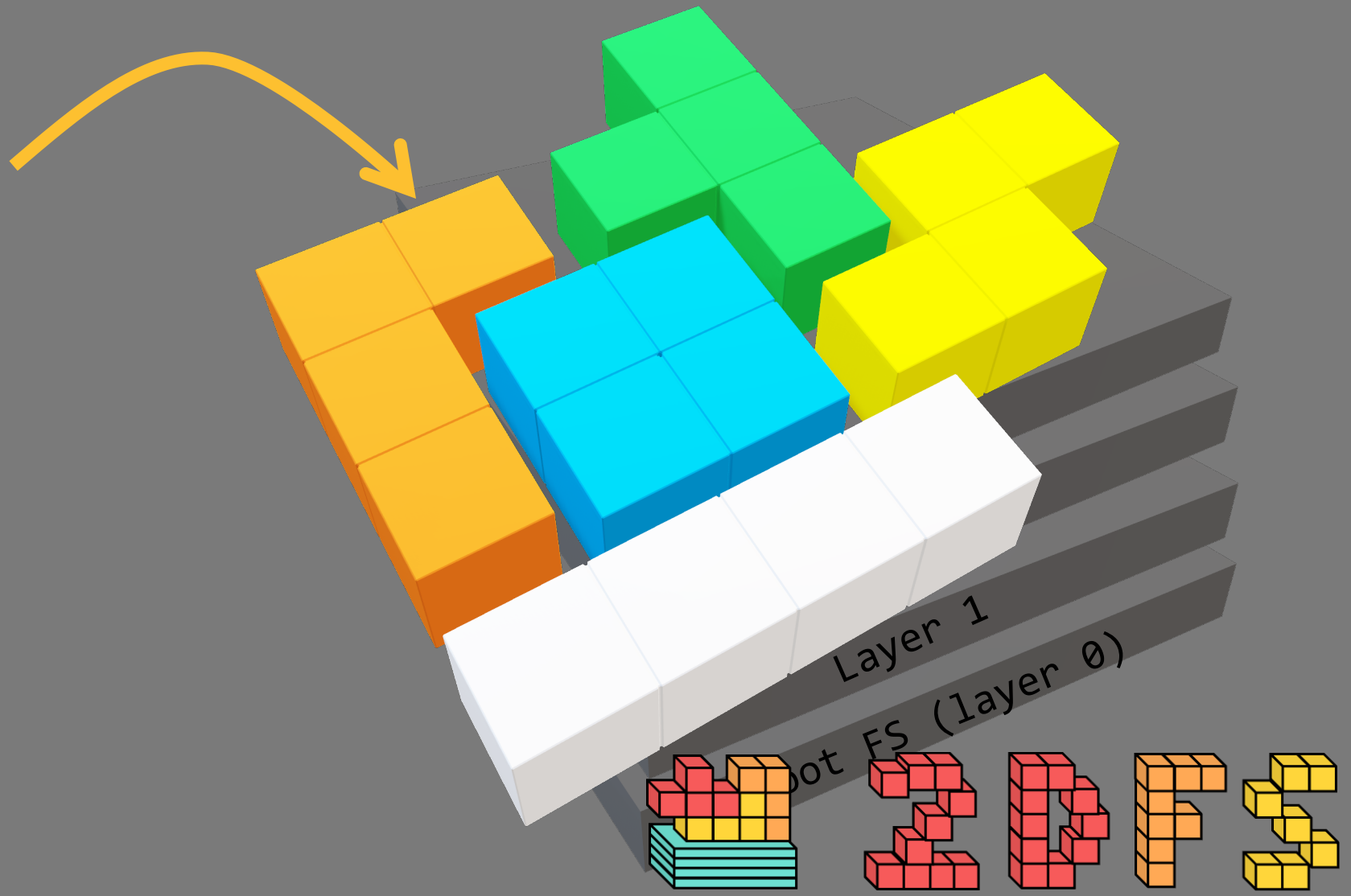
2dfs.field
Is just a new layer type

- This is a sparse matrix that organizes allotments in a grid structure
- Allotments represent a self-contained, non-overlapping, and independent filesystem space
- Each allotment links one or more files or, for instance, a split of a neural network



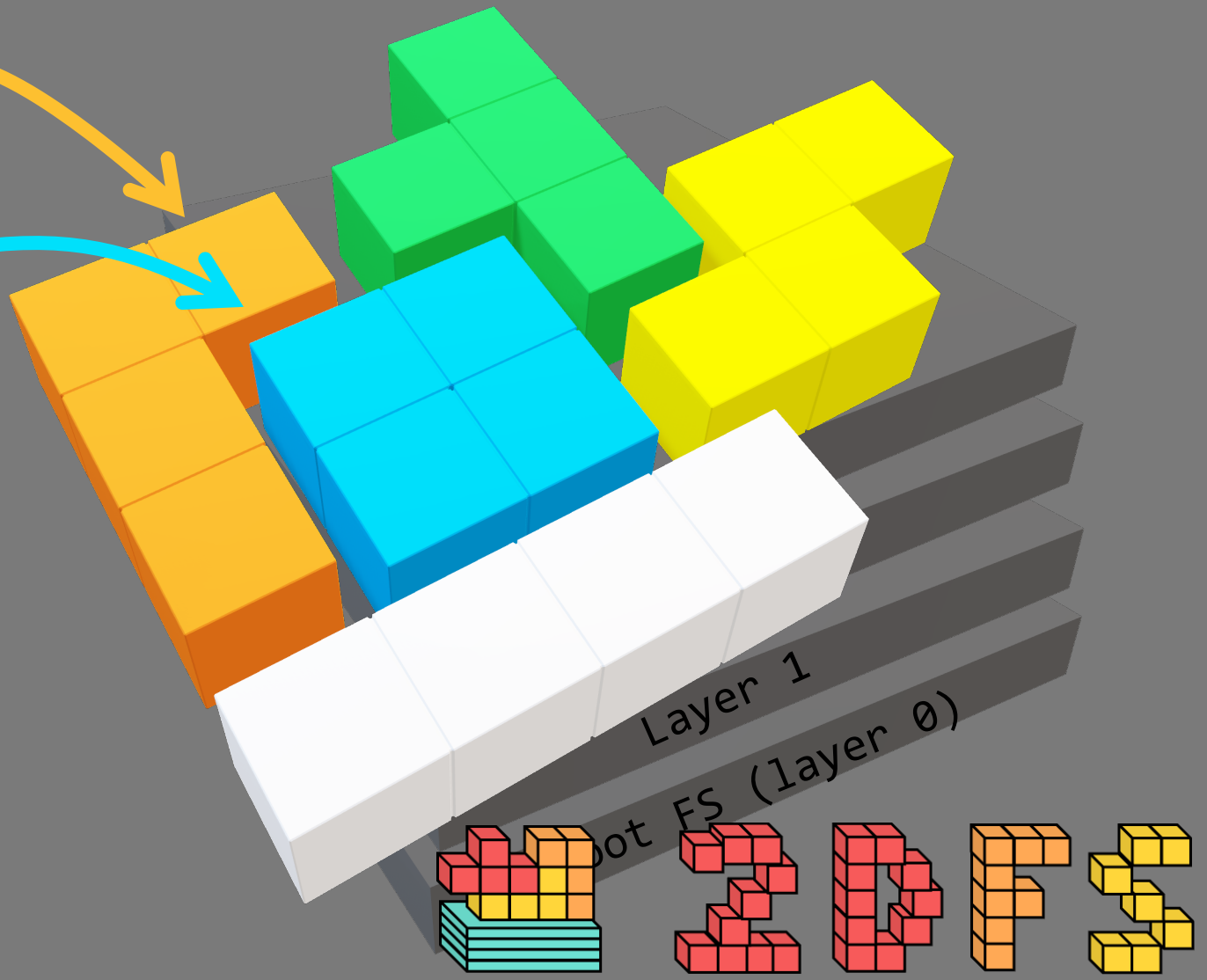
For example:

- Model split 1



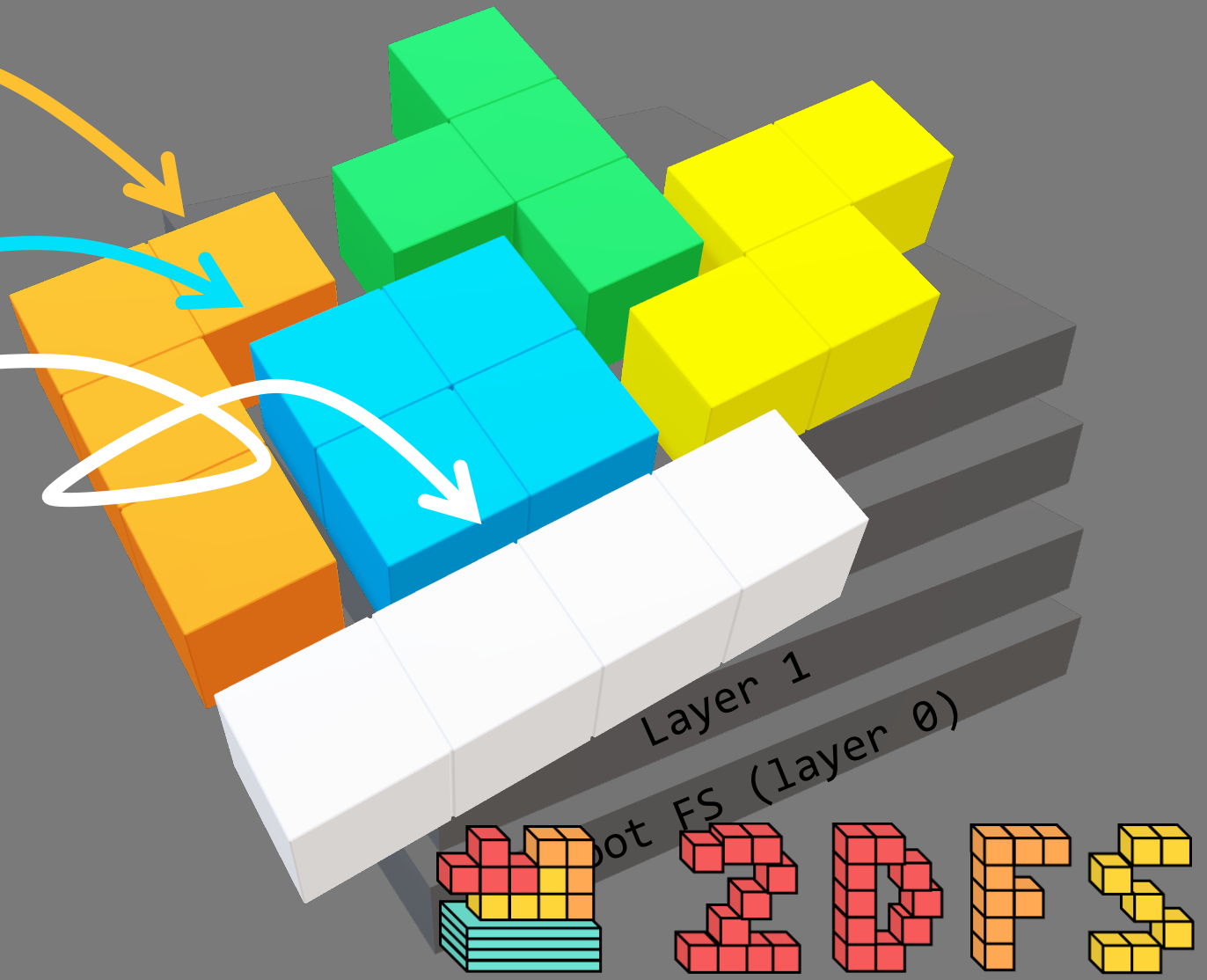
For example:

- Model split 1
- Model Split 2



For example:

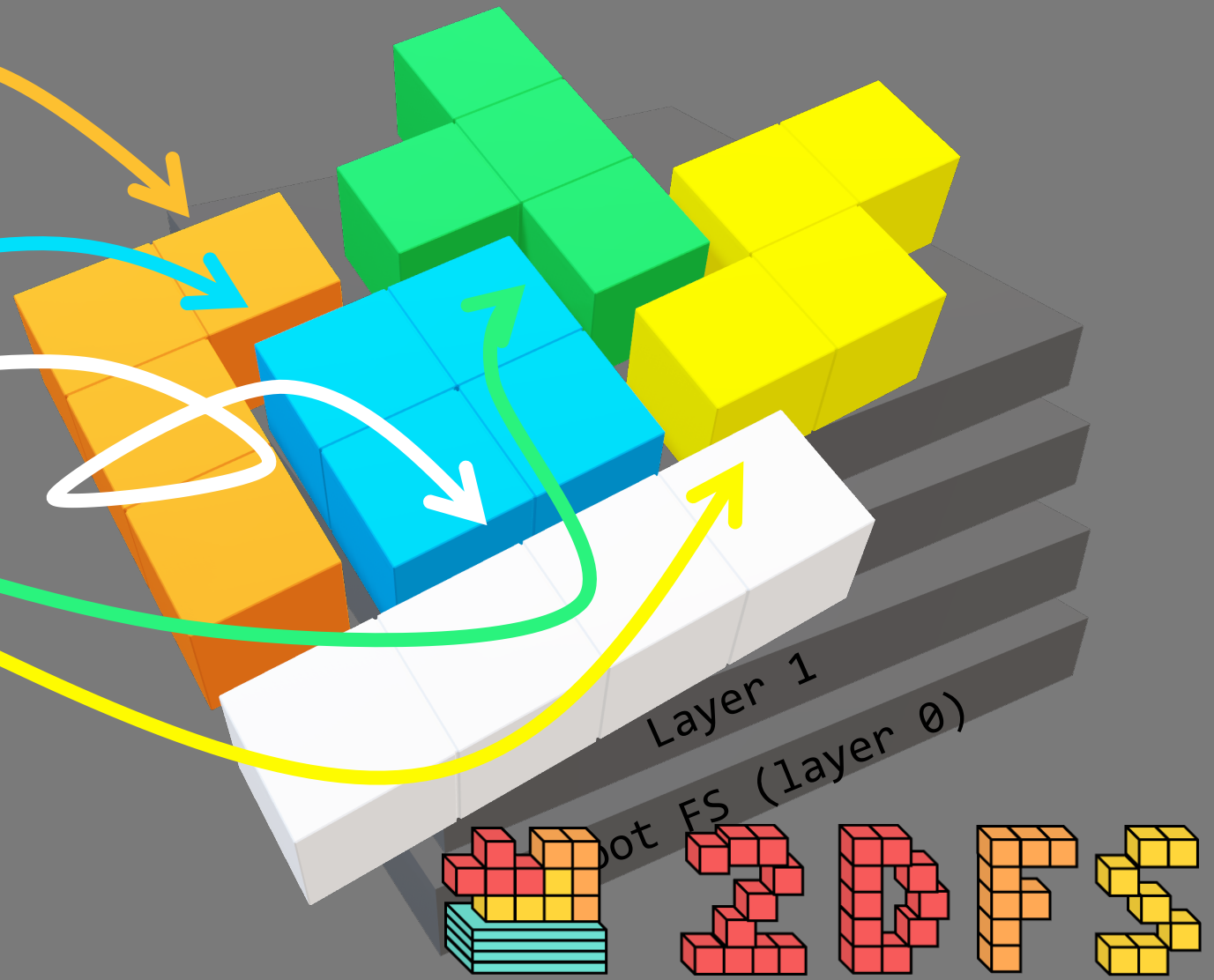
- Model split 1
- Model Split 2
- Model Split 3

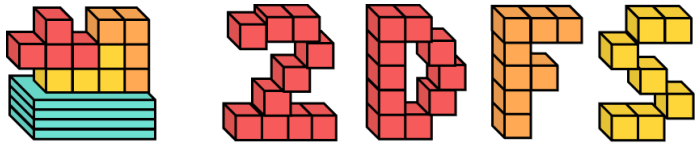


For example:

- Model split 1
- Model Split 2
- Model Split 3
- Nvidia Drivers
- Network Daemon

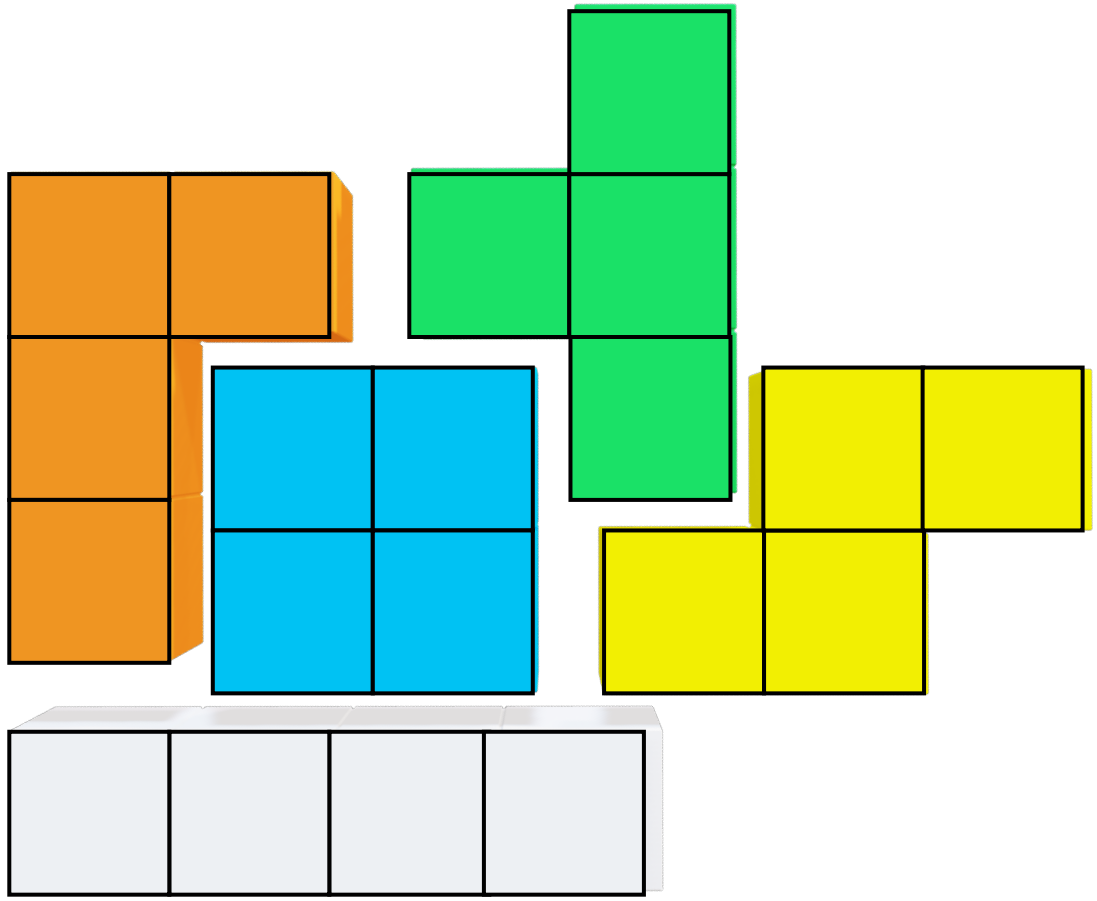
…

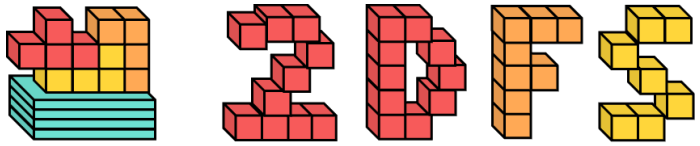




2dfs.field
Is just a new layer type

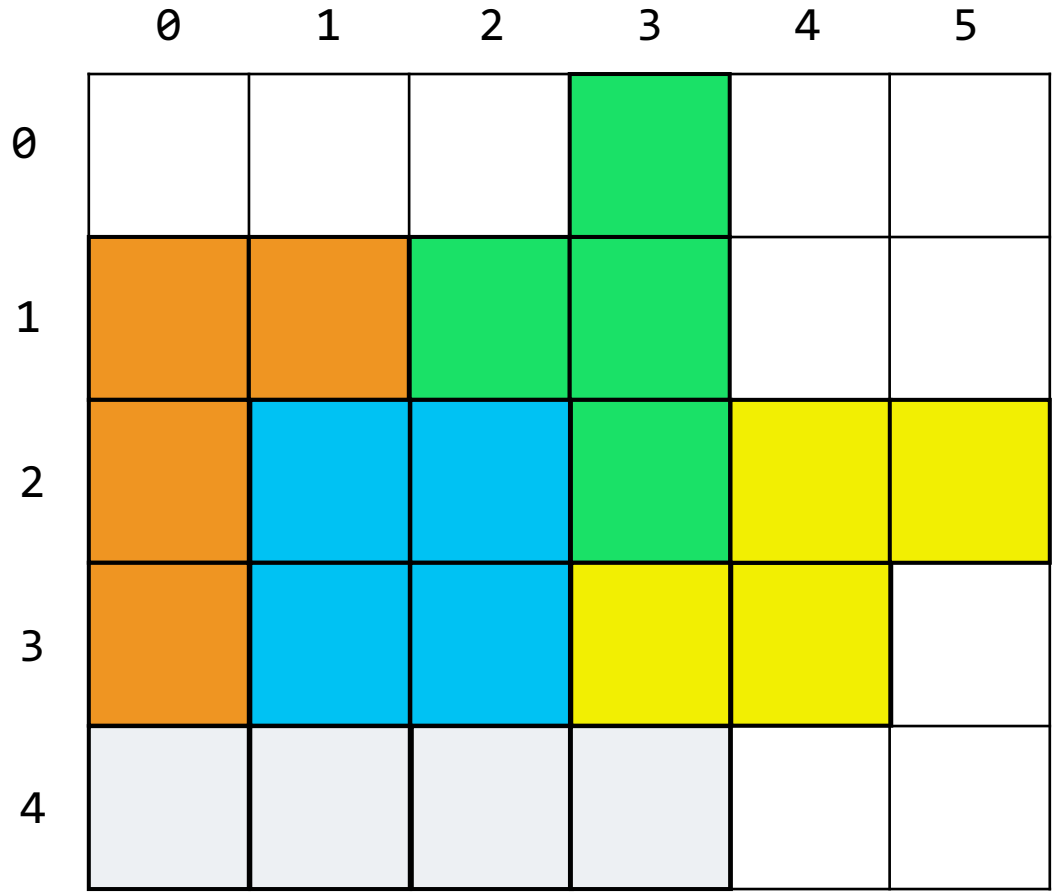
- This is a sparse matrix that organizes allotments in a grid structure
- Allotments represent a self-contained, non-overlapping, and independent filesystem space
- Each allotment links one or more files or, for instance, a split of a neural network

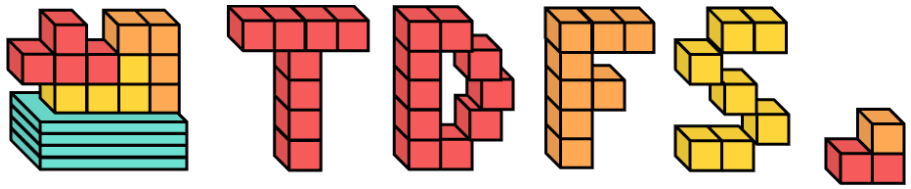




2dfs.field
Is just a new layer type

- This is a sparse matrix that organizes allotments in a grid structure
- Allotments represent a self-contained, non-overlapping, and independent filesystem space
- Each allotment links one or more files or, for instance, a split of a neural network
- Allotments are organized in rows and columns (2D)





tdfs is an OCI+2DFS image builder

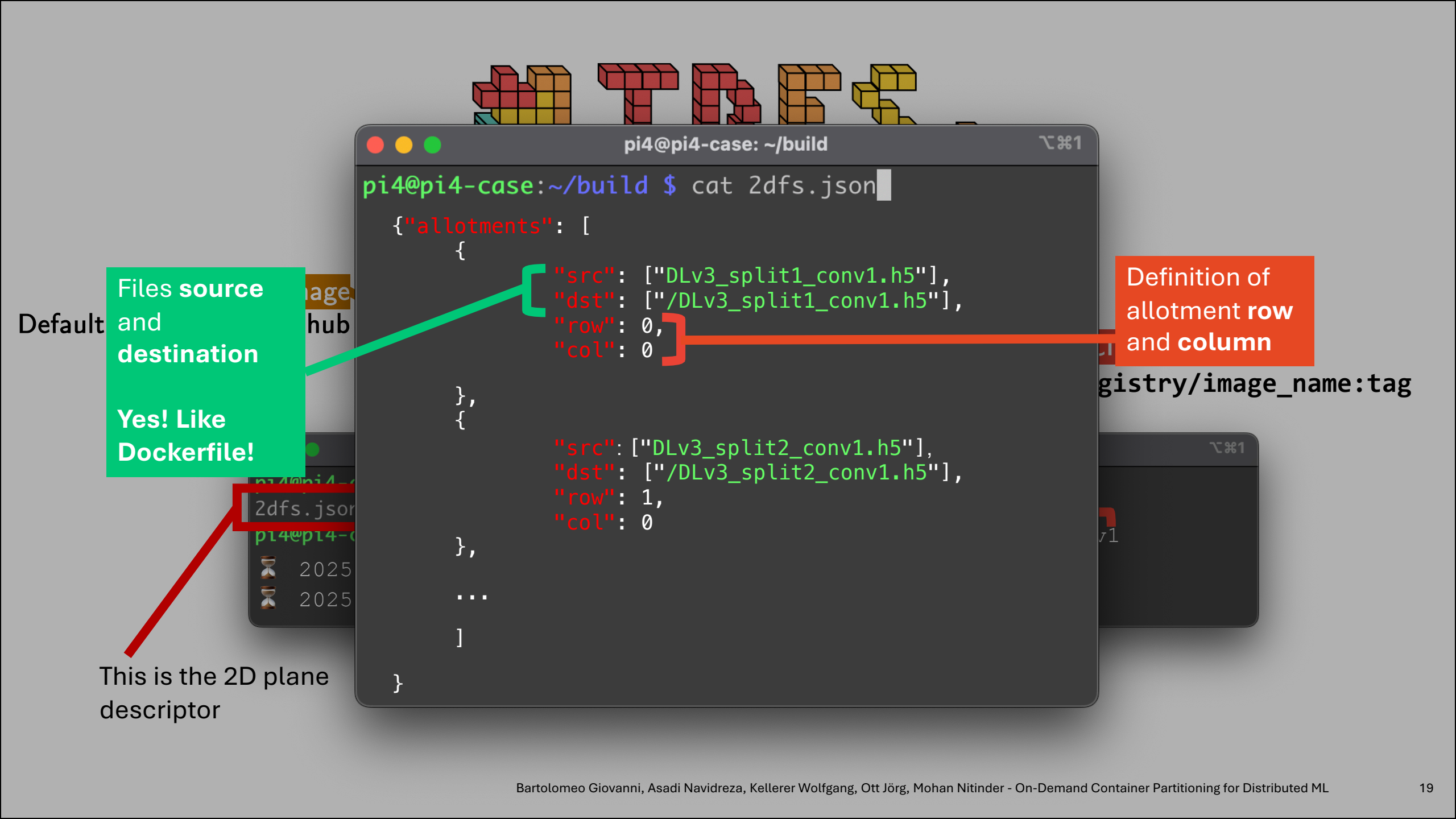
Base OCI image

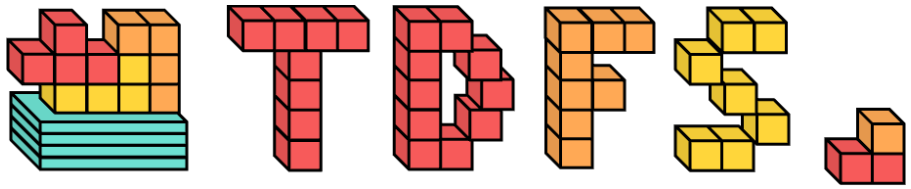
Default registry: docker hub

OCI+2DFS Image

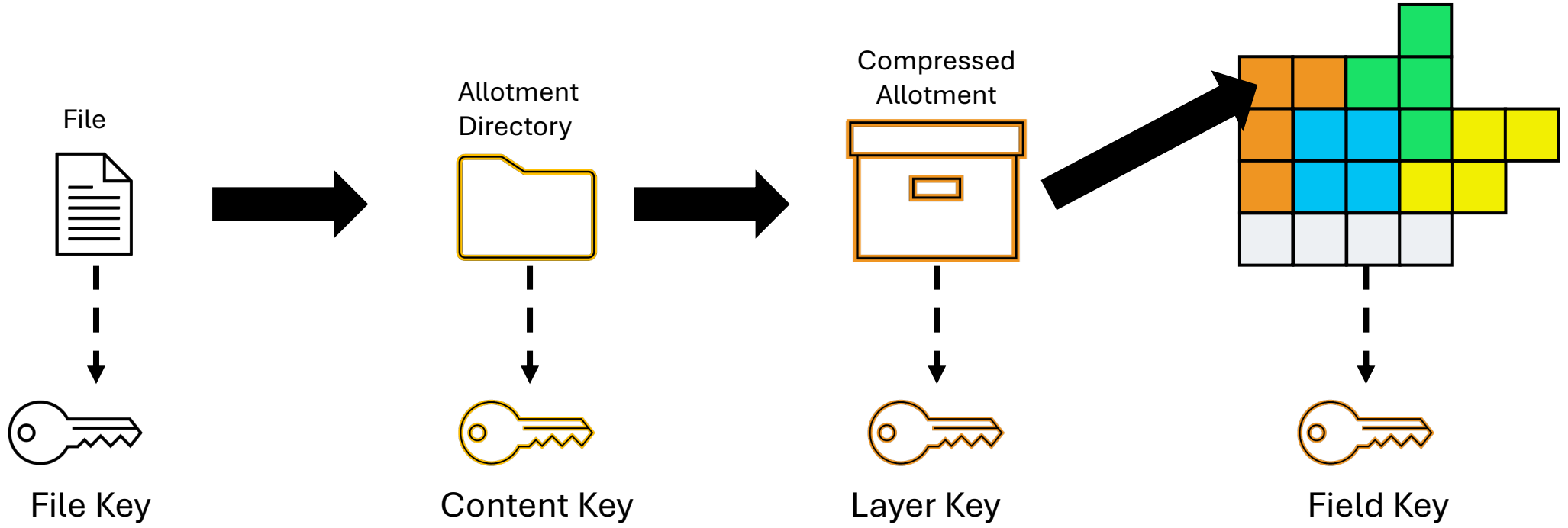
registry/image_name:tag

```
pi4@pi4-case: ~/build                                   ⌥⌘1
pi4@pi4-case:~/build $ ls
2dfs.json
pi4@pi4-case:~/build $ tdfs build ubuntu:22.05 myregistry/image:v1
⏳ 2025/03/06 11:56:08.939845 Build completed ⚒️ (17.410000s)
⏳ 2025/03/06 11:56:08.939849 Done! ✅ (26.820000s)
```

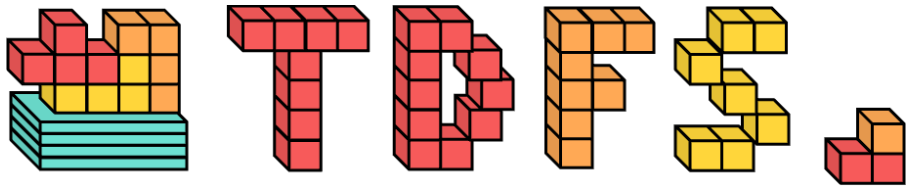

pi4@pi4-case: ~/build
pi4@pi4-case:~/build $ cat 2dfs.json
{"allotments": [
{
"src": ["DLv3_split1_conv1.h5"],
"dst": ["/DLv3_split1_conv1.h5"],
"row": 0,
"col": 0
},
{
"src": ["DLv3_split2_conv1.h5"],
"dst": ["/DLv3_split2_conv1.h5"],
"row": 1,
"col": 0
},
...
]
}
Files source and destination
Yes! Like Dockerfile!
Definition of allotment row and column
Default
hub
gistry/image_name:tag
This is the 2D plane descriptor



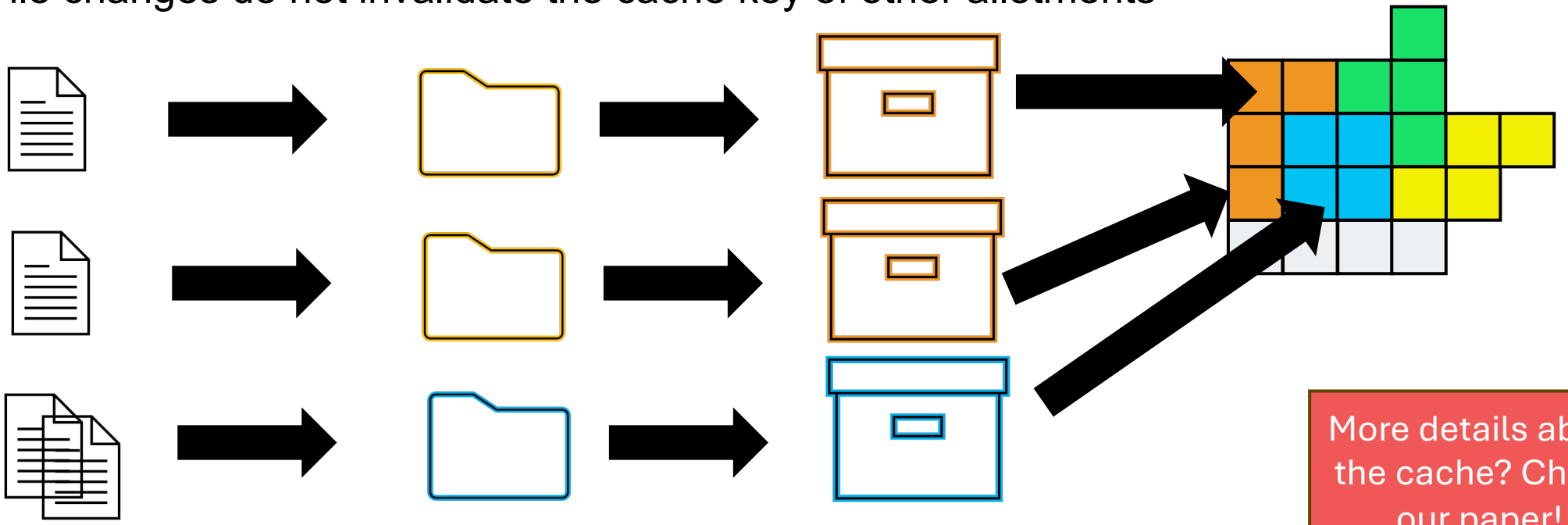
Multi-layer cache system to efficiently store files, allotments, and fields.



- At each build stage the builder caches the result with a cache key
- The builder skips upcoming build stages if a cache key already exists



- All the build stages **are executed in parallel**
- File changes do not invalidate the cache key of other allotments



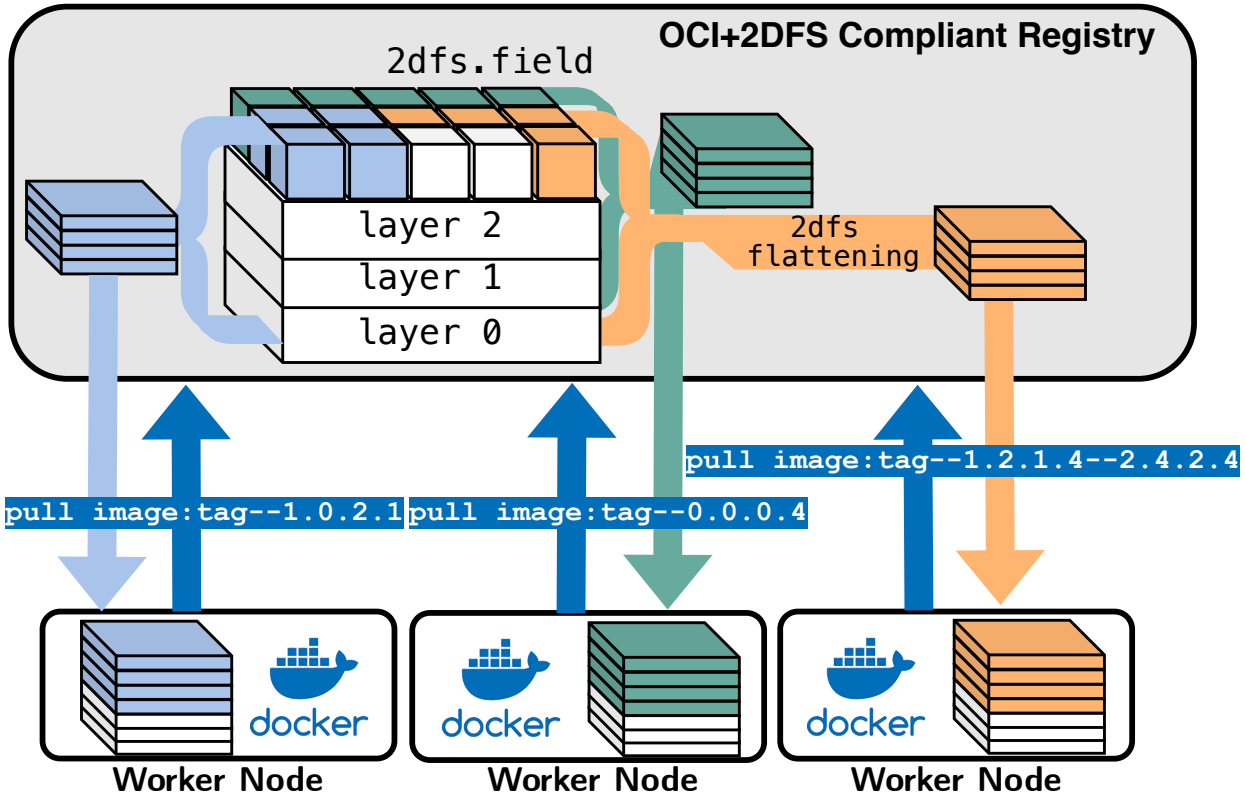
More details about the cache? Check our paper!

Reminder: allotments are self-contained, non-overlapping, and independent filesystem spaces

DISTRIBUTION + 2DFS

OCI+2DFS Registry

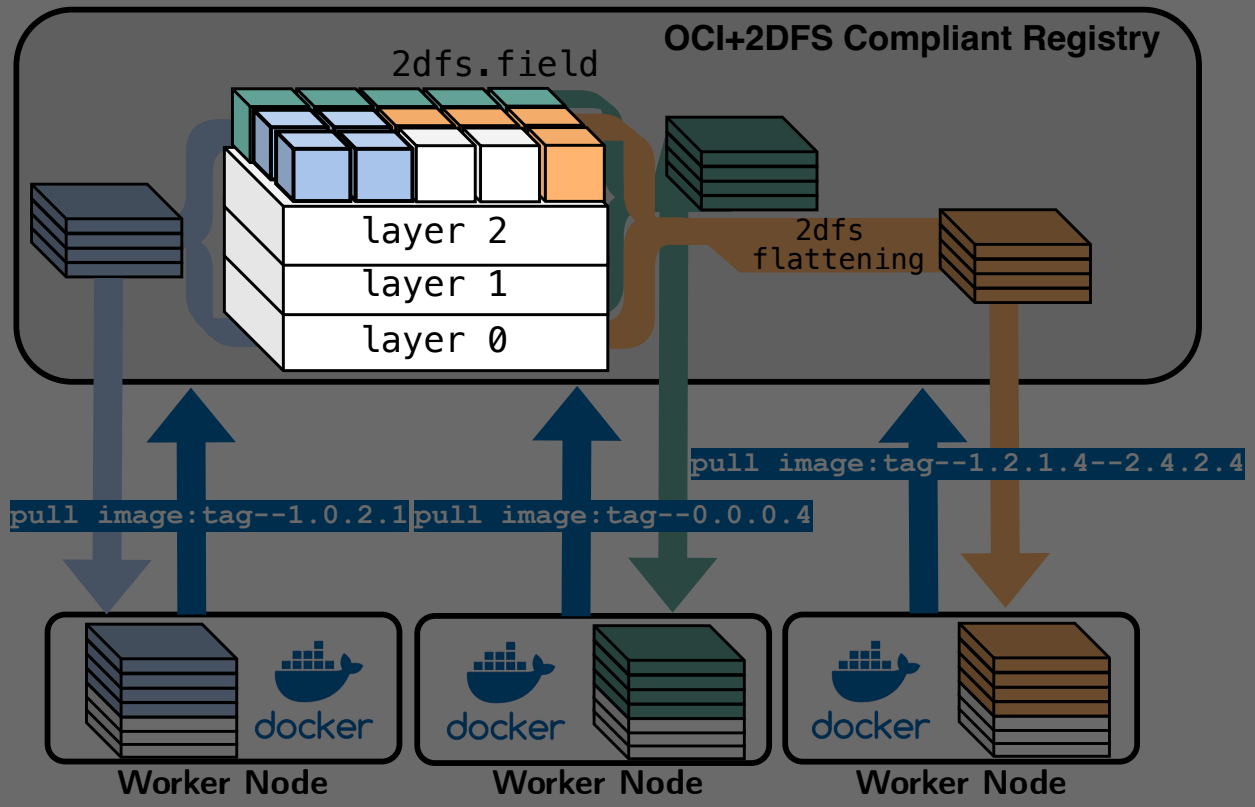
- Container registry to store extended OCI+2DFS images
- Compatibility with `2dfs.field` layer upload and storage
- Semantic tags for on-demand partitioning
- Image flattening for compatibility with pre-existing runtimes



DISTRIBUTION + 2DFS

OCI+2DFS Registry

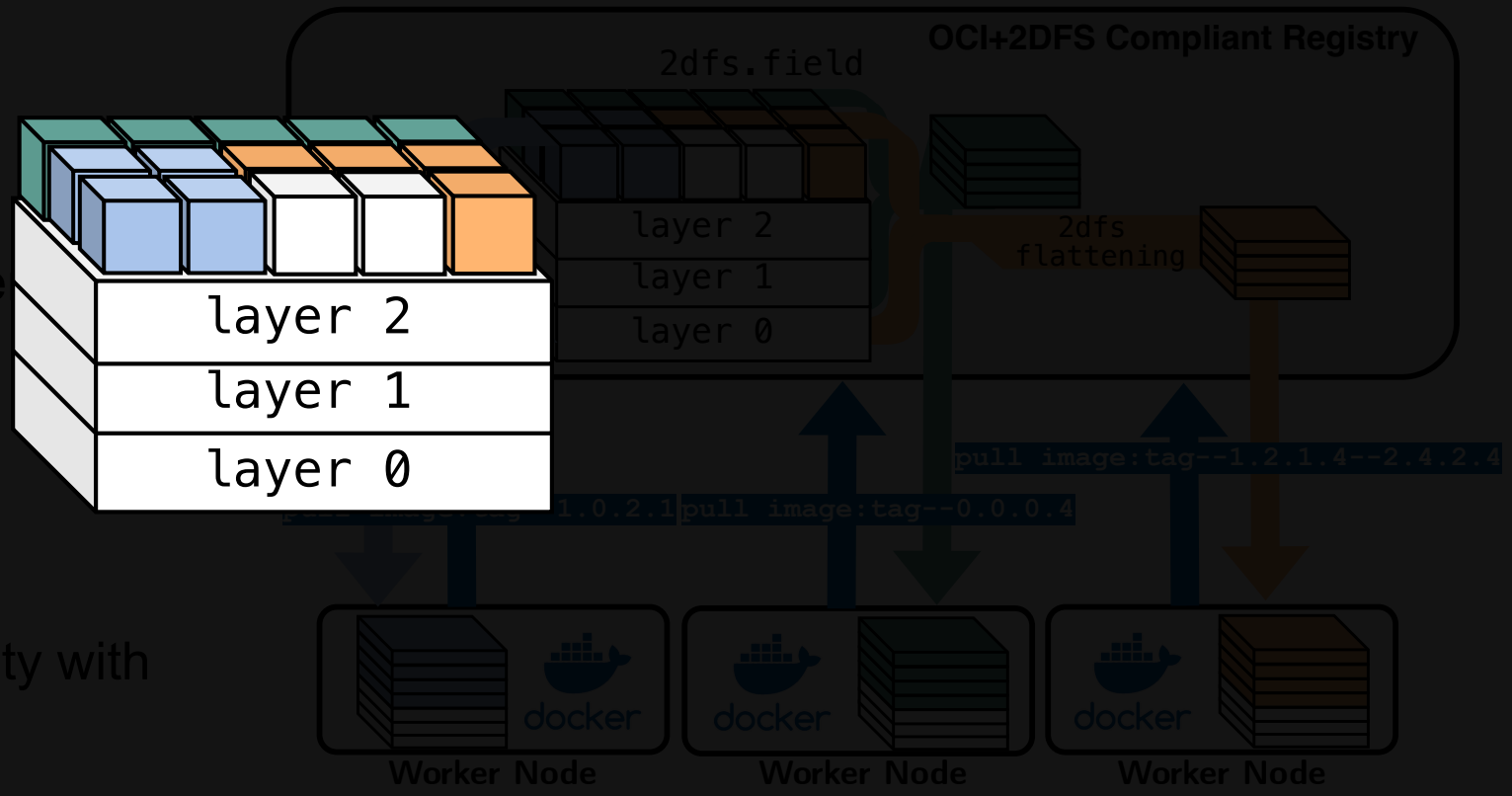
- Container registry to store extended OCI+2DFS images
- Compatibility with `2dfs.field` layer upload and storage
- Semantic tags for on-demand partitioning
- Image flattening for compatibility with pre-existing runtimes



DISTRIBUTION + 2DFS

OCI+2DFS Registry

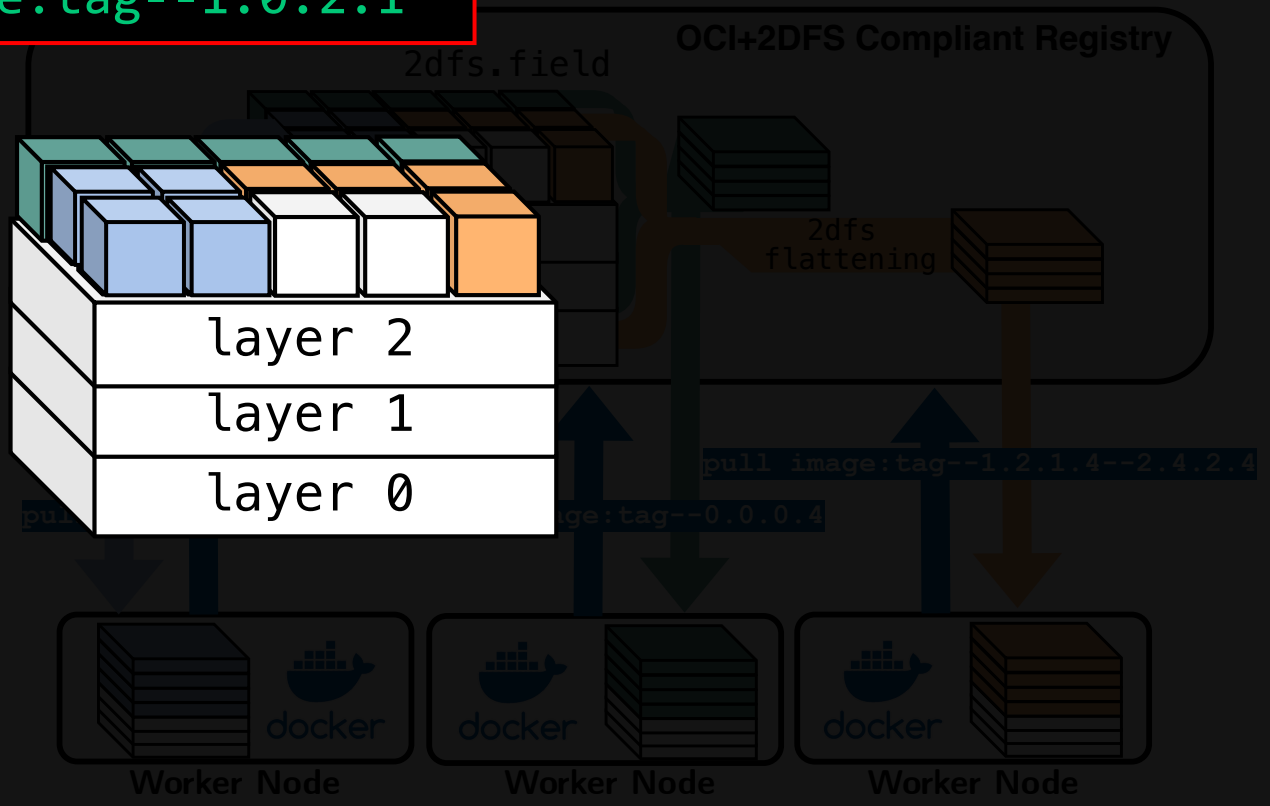
- Container registry to store exte OCI+2DFS images
- Compatibility with `2dfs.field` upload and storage
- Semantic tags for on-demand partitioning
- Image flattening for compatibility with pre-existing runtimes



DISTRIBUTION + 2DFS

```
docker pull registry/image:tag--1.0.2.1
```

Select allotments
from: row 1 col 0
to: row 2 col 1



OCI+2DFS Registry

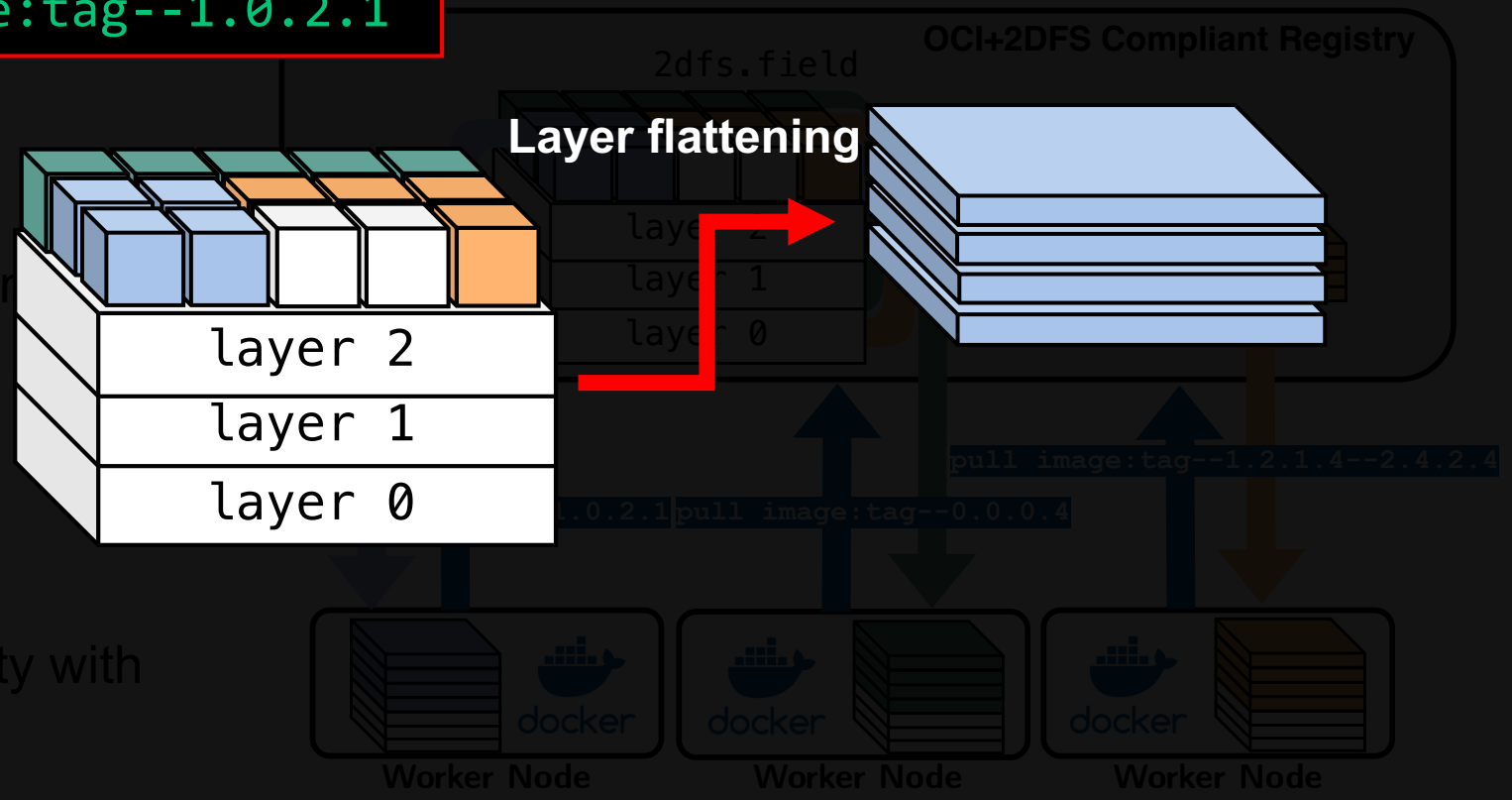
- Container registry to store extended OCI+2DFS images
- Compatibility with 2dfs.field layer upload and storage
- Semantic tags for on-demand partitioning
- Image flattening for compatibility with pre-existing runtimes

DISTRIBUTION + 2DFS

```
docker pull registry/image:tag--1.0.2.1
```

OCI+2DFS Registry

- Container registry to store extended OCI+2DFS images
- Compatibility with `2dfs.field` upload and storage
- Semantic tags for on-demand partitioning
- Image flattening for compatibility with pre-existing runtimes



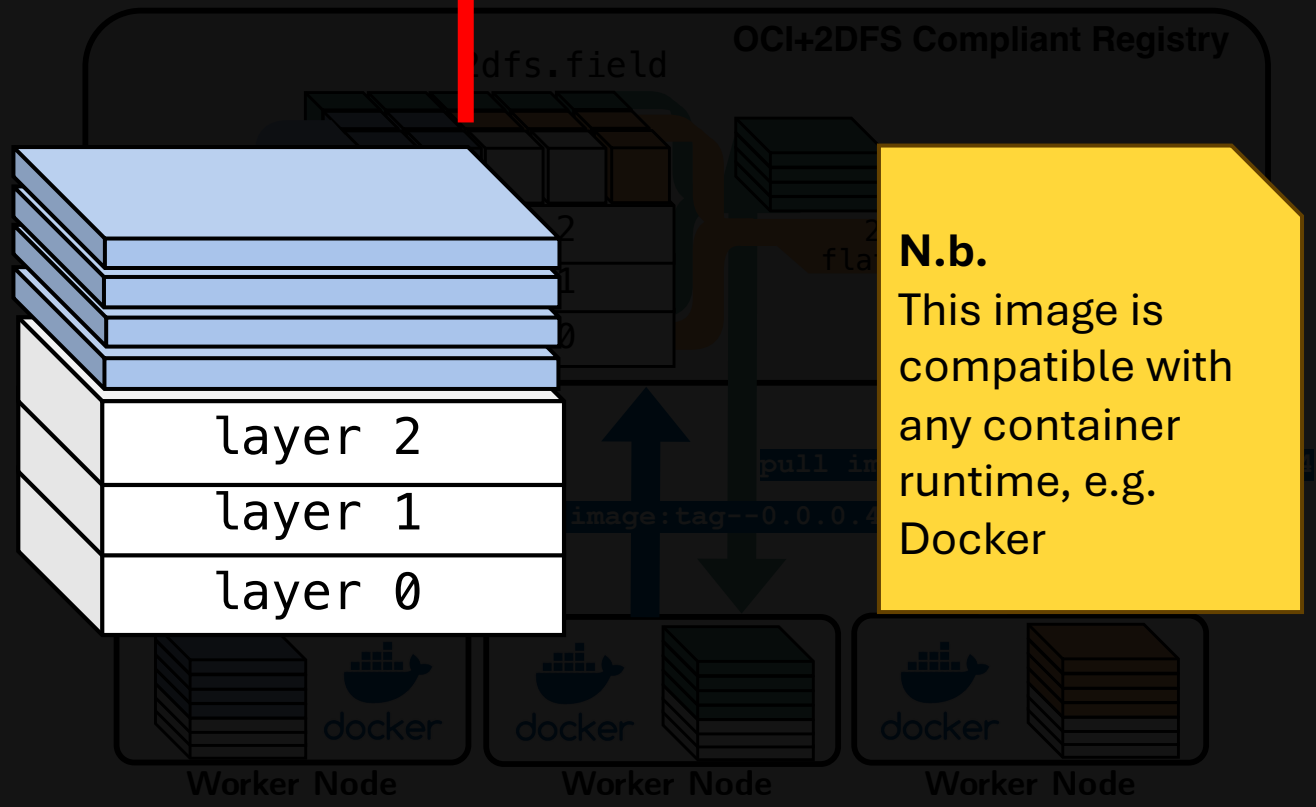
```
docker pull registry/image:tag--1.0.2.1
```

Regular OCI Image

Select allotments
from: `row 1 col 0`
to: `row 2 col 1`

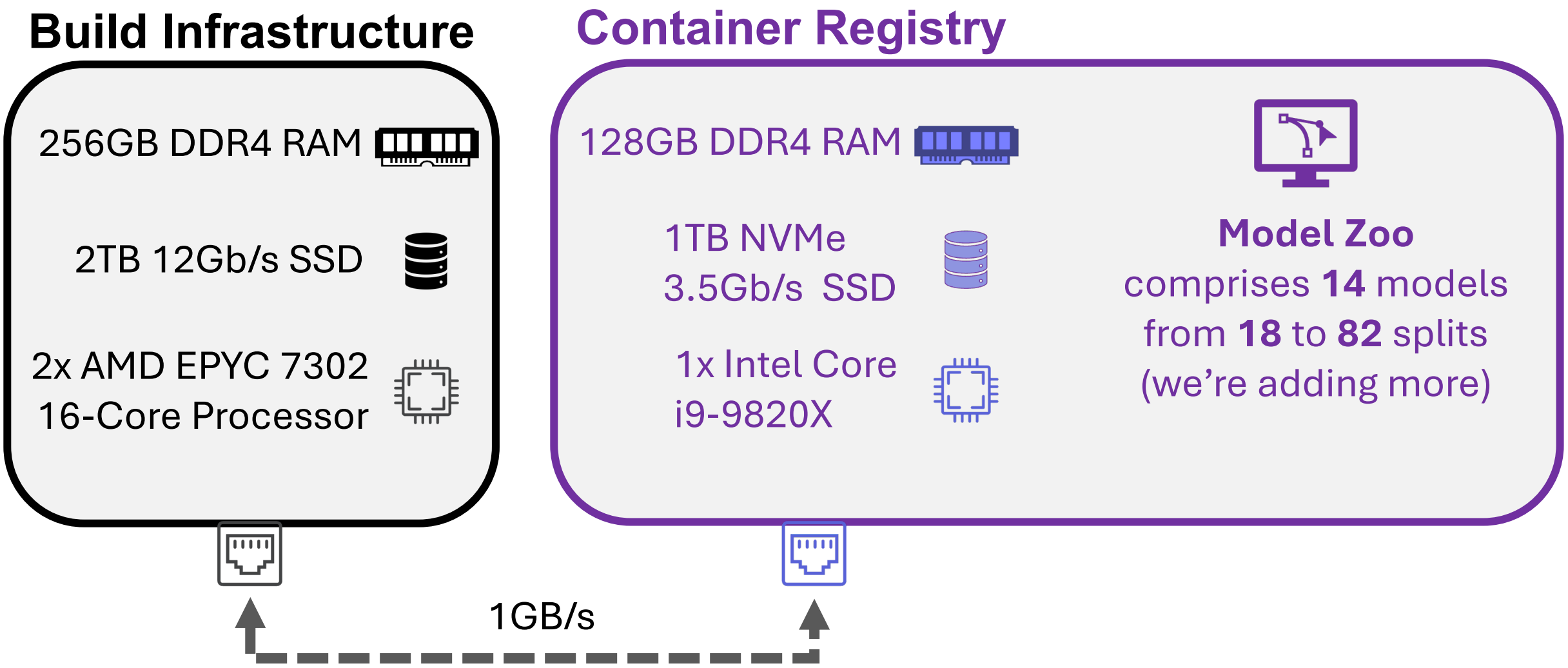
OCI+2DFS Registry

- Container registry to store extended OCI+2DFS images
- Compatibility with `2dfs.field` layer upload and storage
- Semantic tags for on-demand partitioning
- Image flattening for compatibility with pre-existing runtimes



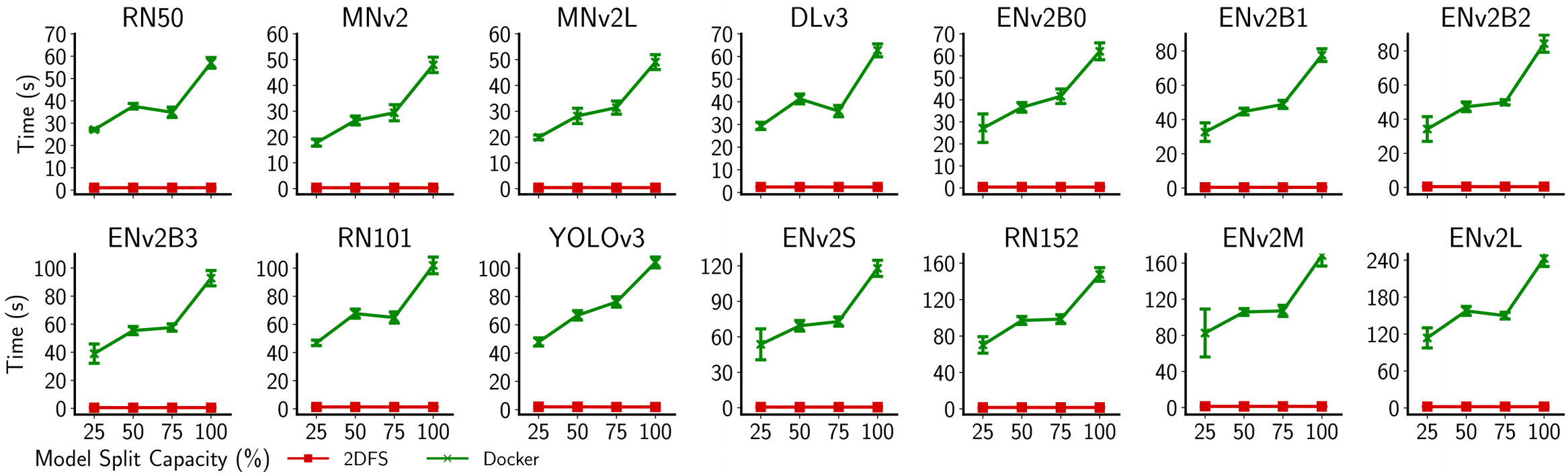
N.b.
This image is compatible with any container runtime, e.g. Docker

Evaluation


Build Infrastructure
256GB DDR4 RAM
2TB 12Gb/s SSD
2x AMD EPYC 7302
16-Core Processor
Container Registry
128GB DDR4 RAM
1TB NVMe
3.5Gb/s SSD
1x Intel Core
i9-9820X
Model Zoo
comprises 14 models
from 18 to 82 splits
(we're adding more)
1GB/s

Build time for different split partitions

2DFS goes brrr



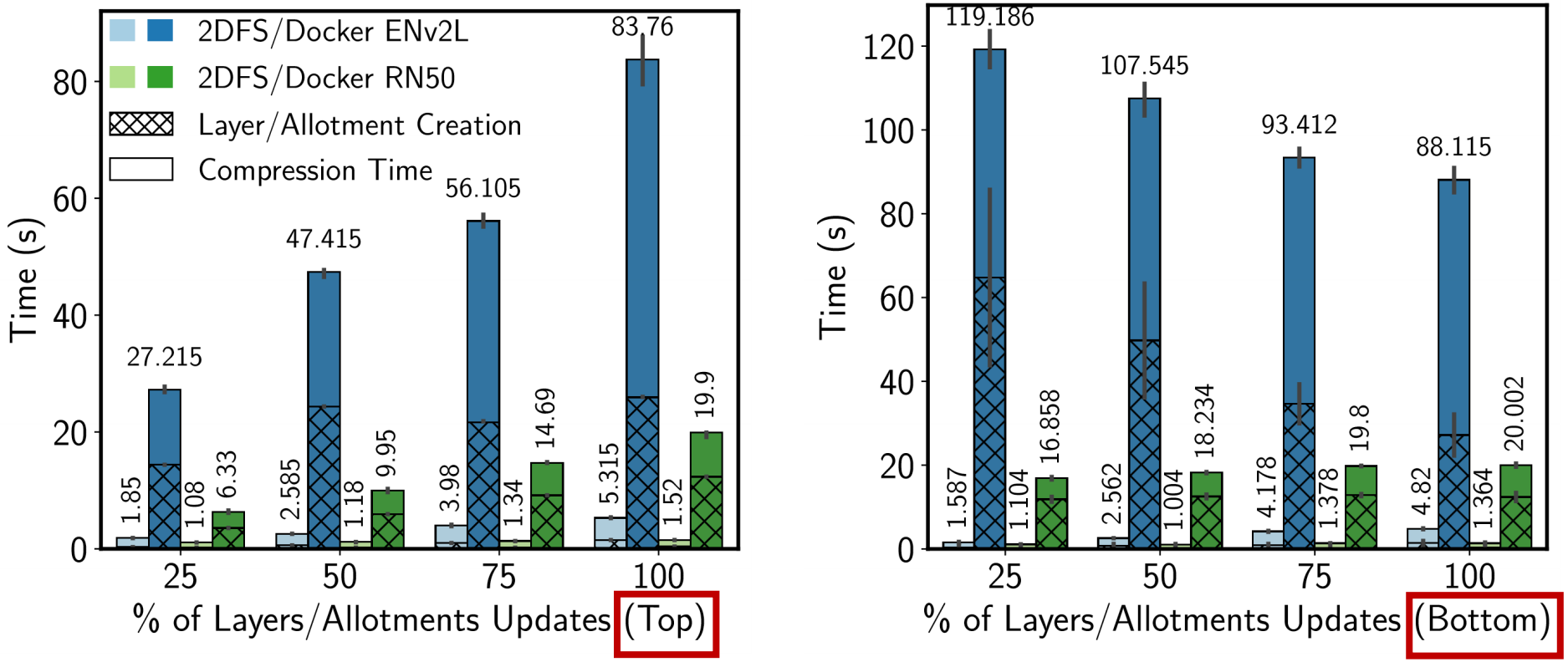
56x faster build time on avg compared to traditional Docker build

120x fastest build time peak

More details?
Check our paper!

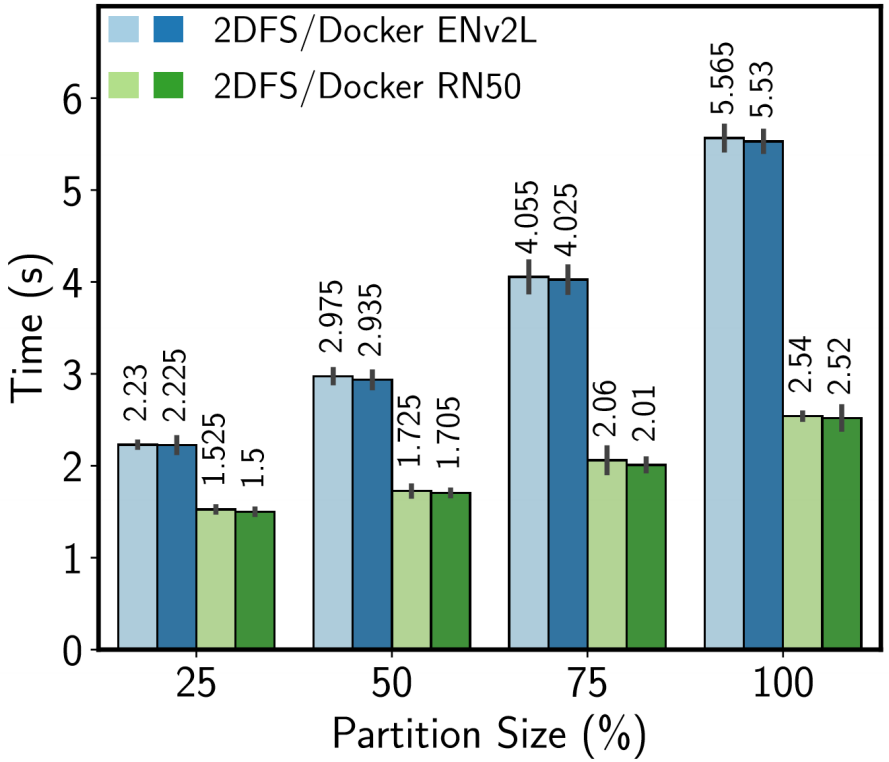
Model Updates
2DFS vs. Optimized Docker

Lower, Better!



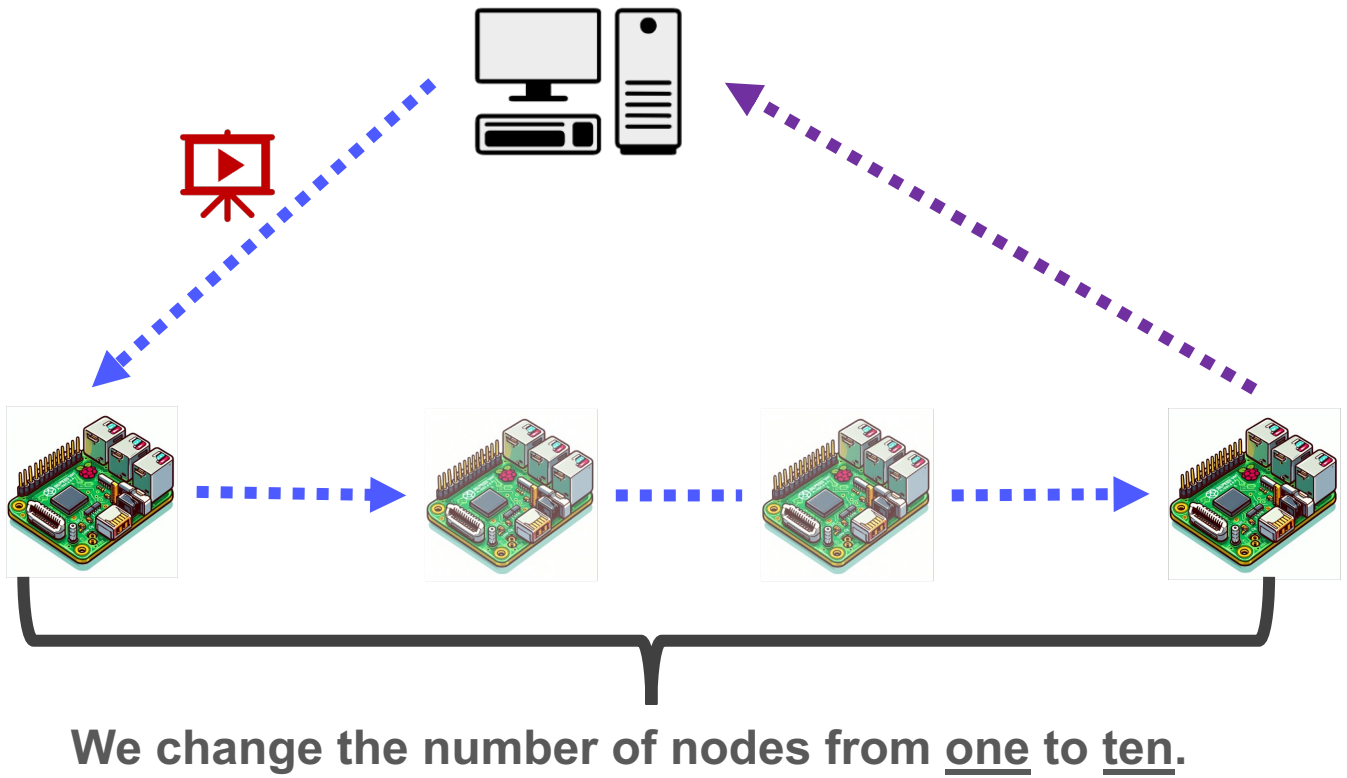
25x improved caching efficiency in top -> bottom scenario om average
75x improved caching efficiency in bottom -> up update scenario om average

2DFS and Docker have comparable download time



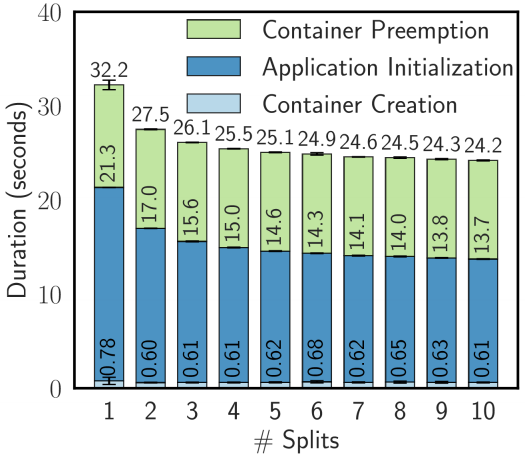
Allotments flattening accounts on average for additional 20ms overhead in download time

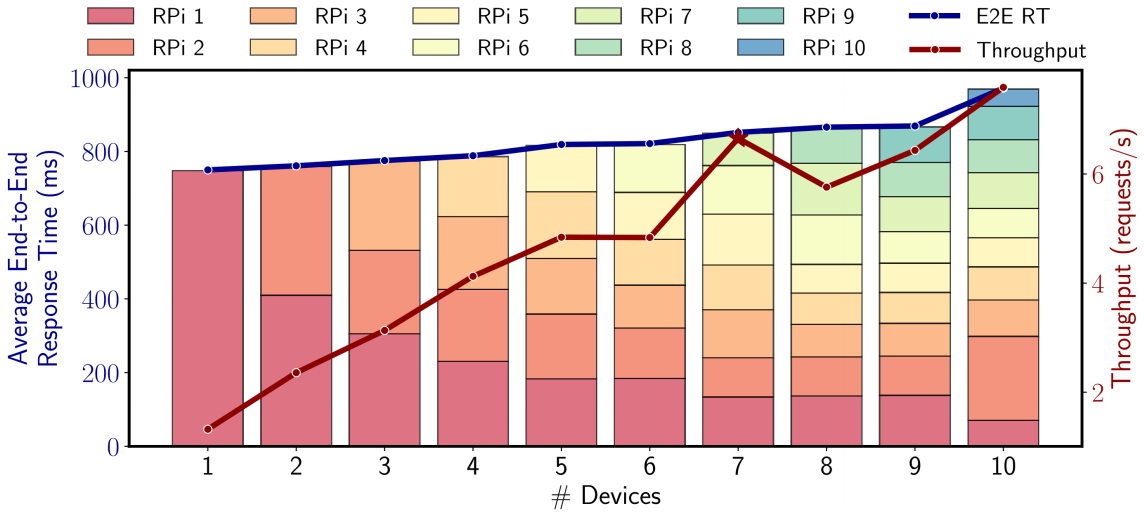
A working example - Split Computing on 1, 2, …, up to 10 Edge Nodes

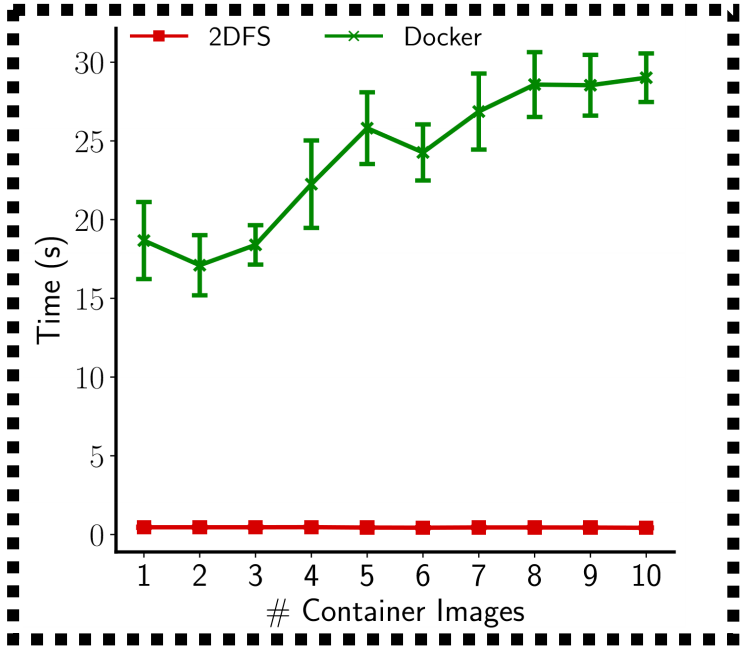


- Async non-blocking streams
- Use gRPC for intercommunication
- A PC acts as both the source of data generation and the sink
  - The PC runs an NTP server
  - Distributed Mobile Net

A working example - Split Computing on 1, 2, …, up to 10 Edge Nodes







Build time for different number of splits/contributing nodes

Questions?

On-Demand Container Partitioning for Distributed ML

Giovanni Bartolomeo[#*]
Navidreza Asadi[#*]
Wolfgang Kellerer[#]
Jörg Ott[#]
Nitinder Mohan[Δ]

Technical University of Munich

TU Delft Delft University of Technology

Anyone interested in helping us host 2DFS Container Registry?

* Equal Contribution
#Technical University of Munich, Germany, {name.surname}@tum.de
ΔTU Delft, The Netherlands, n.mohan@tudelft.nl

6G-life

FUTURE 6G NETWORK SERVICES