[544] Distributed Computing

Tyler Caraza-Harter

Learning Objectives

- configure SSH tunneling and Docker port forwarding to communicate with an app in a container on a different machine
- deploy multi-container apps with Docker compose
- identify situations where replication and/or some variant of partitioning is useful

Outline

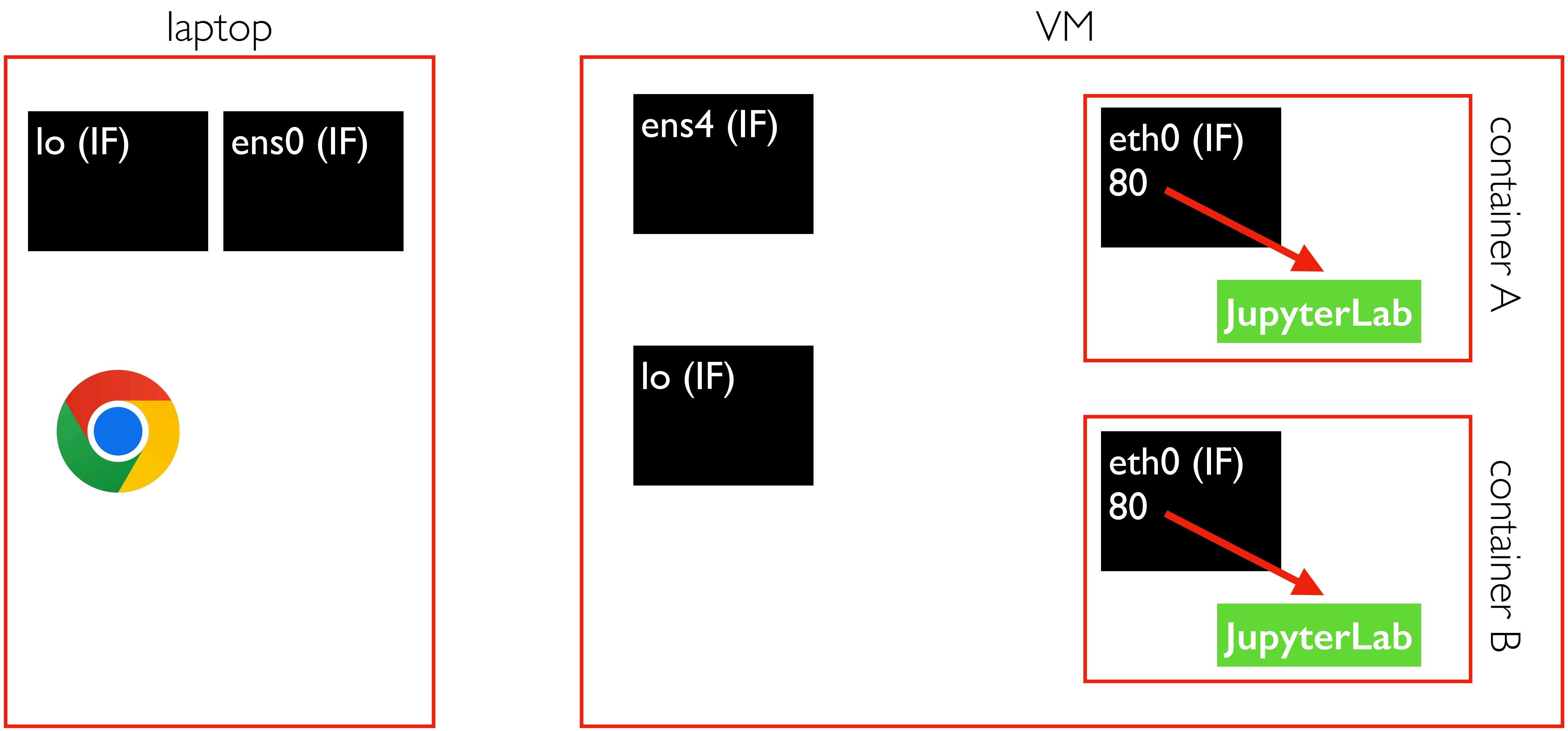
Docker Port Forwarding

Docker Compose

Partitioning and Replication

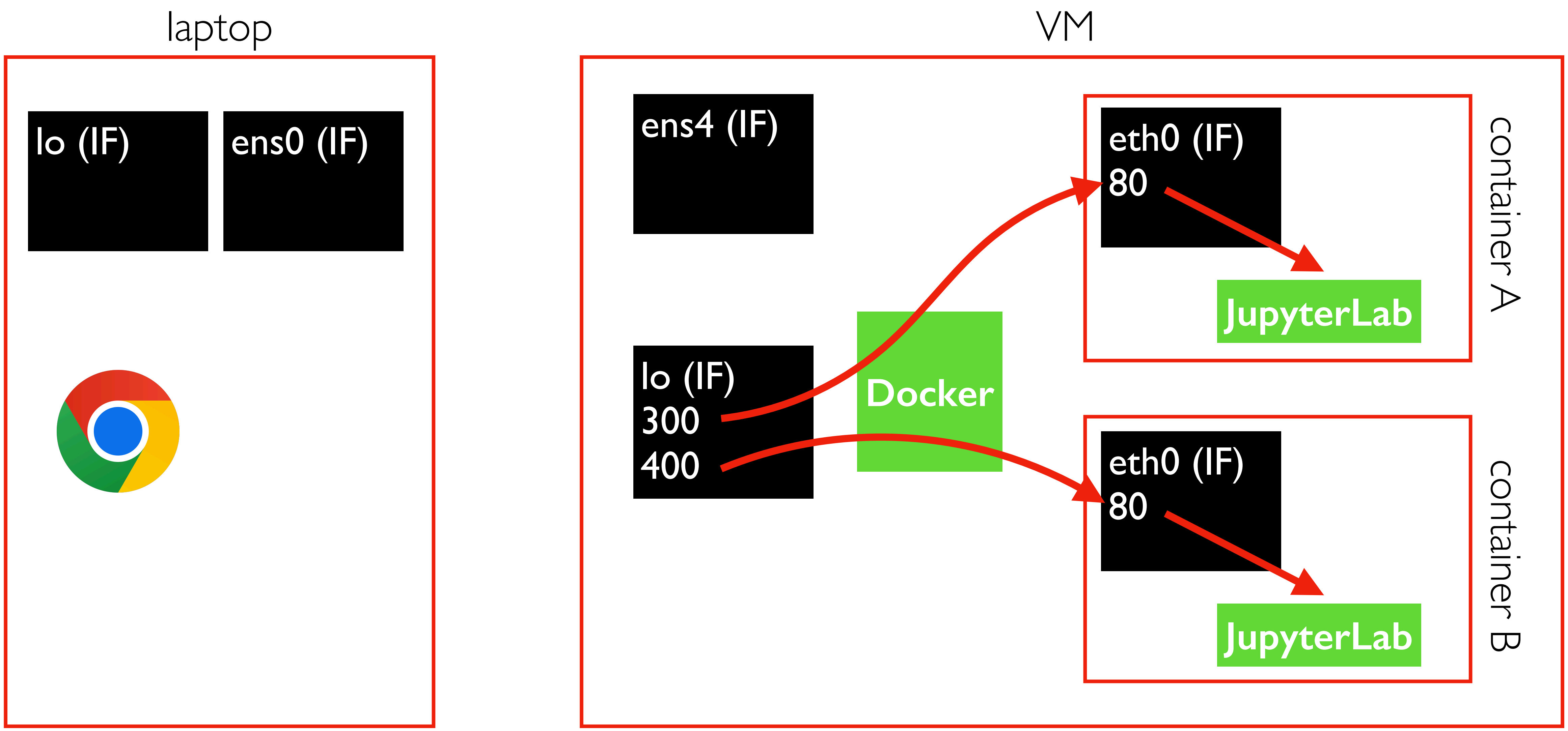
Interfaces (IF) and Ports

both containers have
a virtual port 80



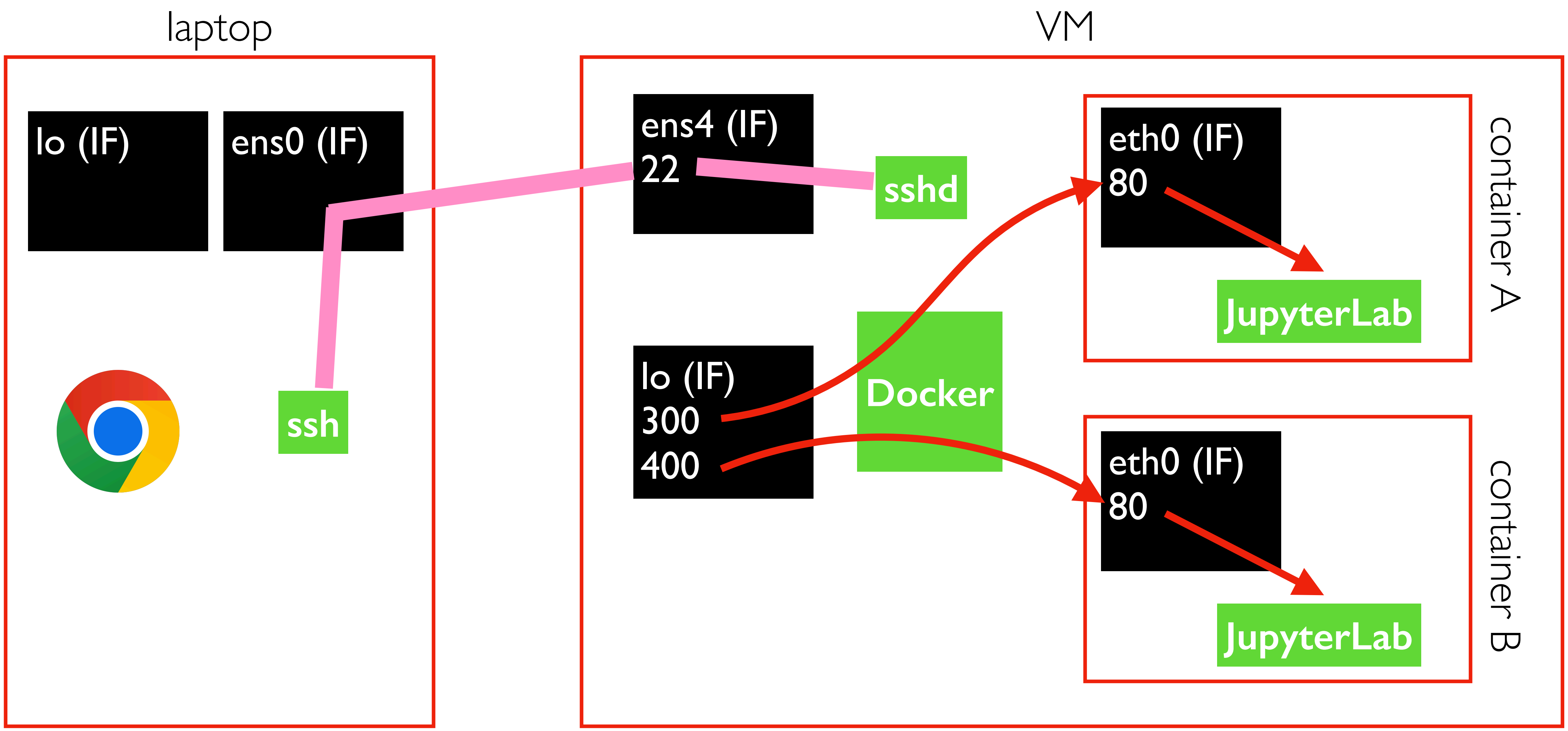

```
docker run -d myimg
docker run -d myimg
```

Interfaces (IF) and Ports




```
docker run -d -p 127.0.0.1:300:80 myimg
docker run -d -p 127.0.0.1:400:80 myimg
```

Interfaces (IF) and Ports

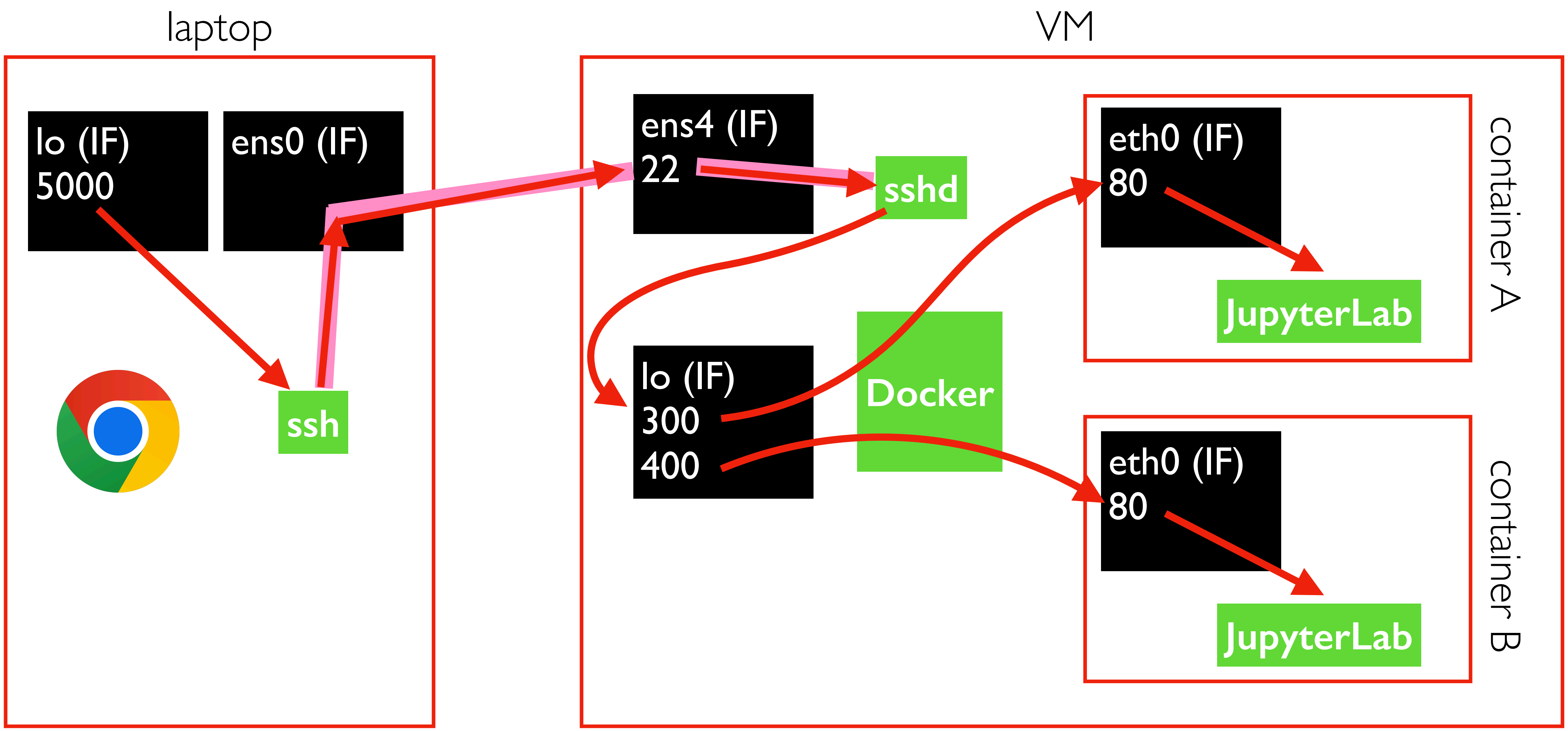


ssh USER@VM

docker run -d **-p** 127.0.0.1:300:80 myimg
docker run -d **-p** 127.0.0.1:400:80 myimg

the SSH connection can be used to send comands and/or forward network traffic

Interfaces (IF) and Ports

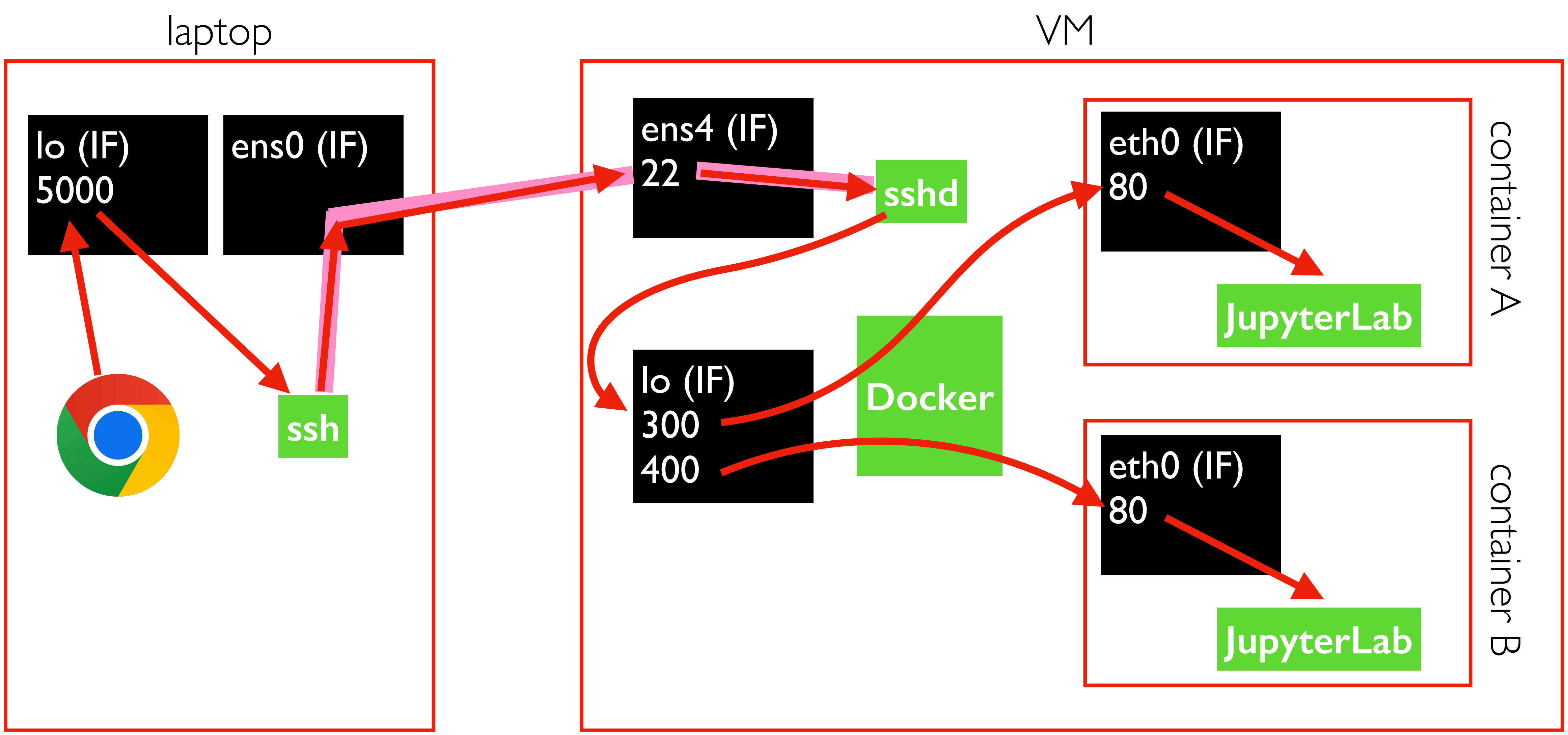


ssh USER@VM **-L** localhost:5000:localhost:300

docker run -d **-p** 127.0.0.1:300:80 myimg
docker run -d **-p** 127.0.0.1:400:80 myimg

the SSH connection can be used to send comands and/or forward network traffic

Interfaces (IF) and Ports



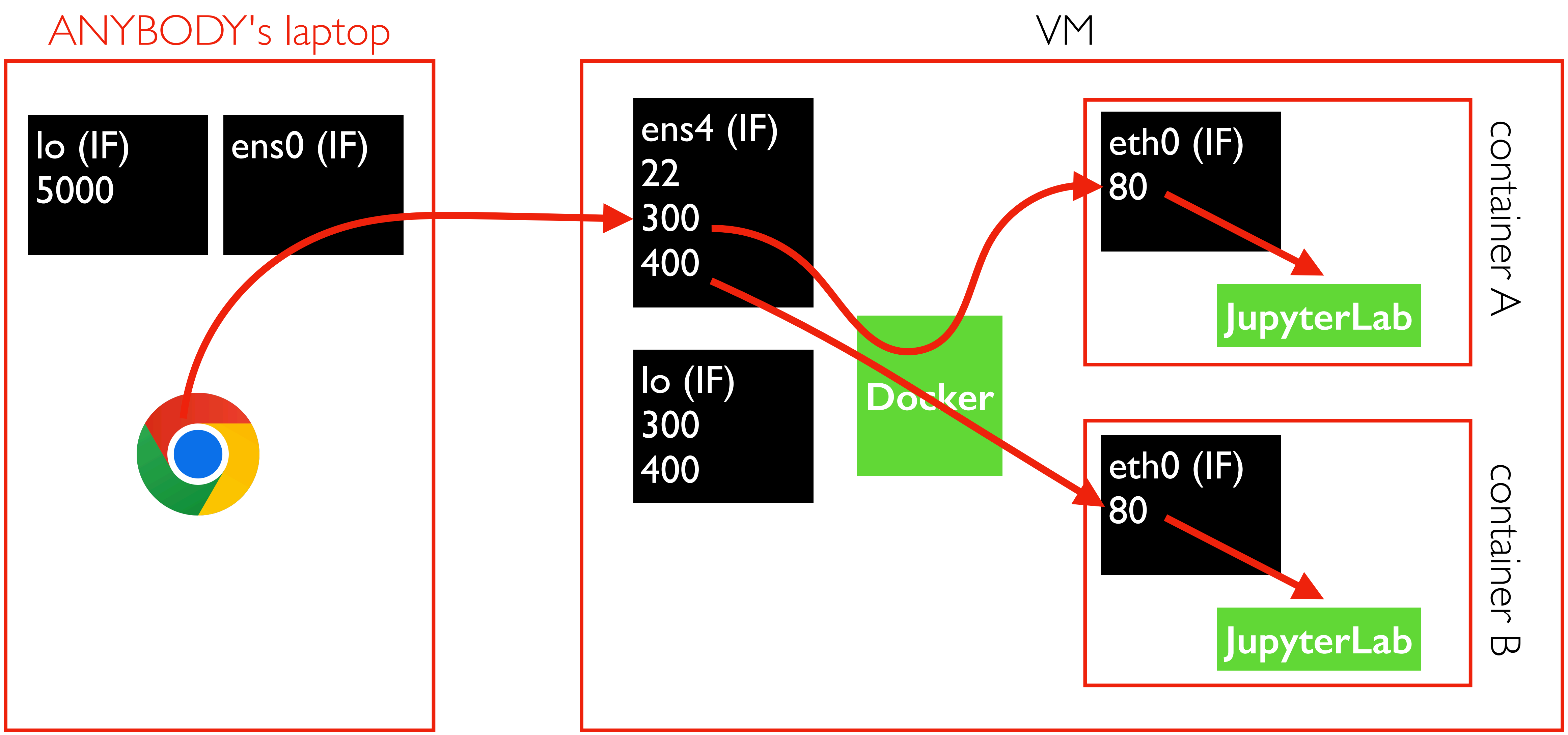
ssh USER@VM **-L** localhost:5000:localhost:300

docker run -d **-p** 127.0.0.1:300:80 myimg
docker run -d **-p** 127.0.0.1:400:80 myimg

http://localhost:5000/lab (in browser)

yay! You can connect to JupyterLab inside a container running on your VM

Interfaces (IF) and Ports



docker run -d **-p** 300:80 myimg

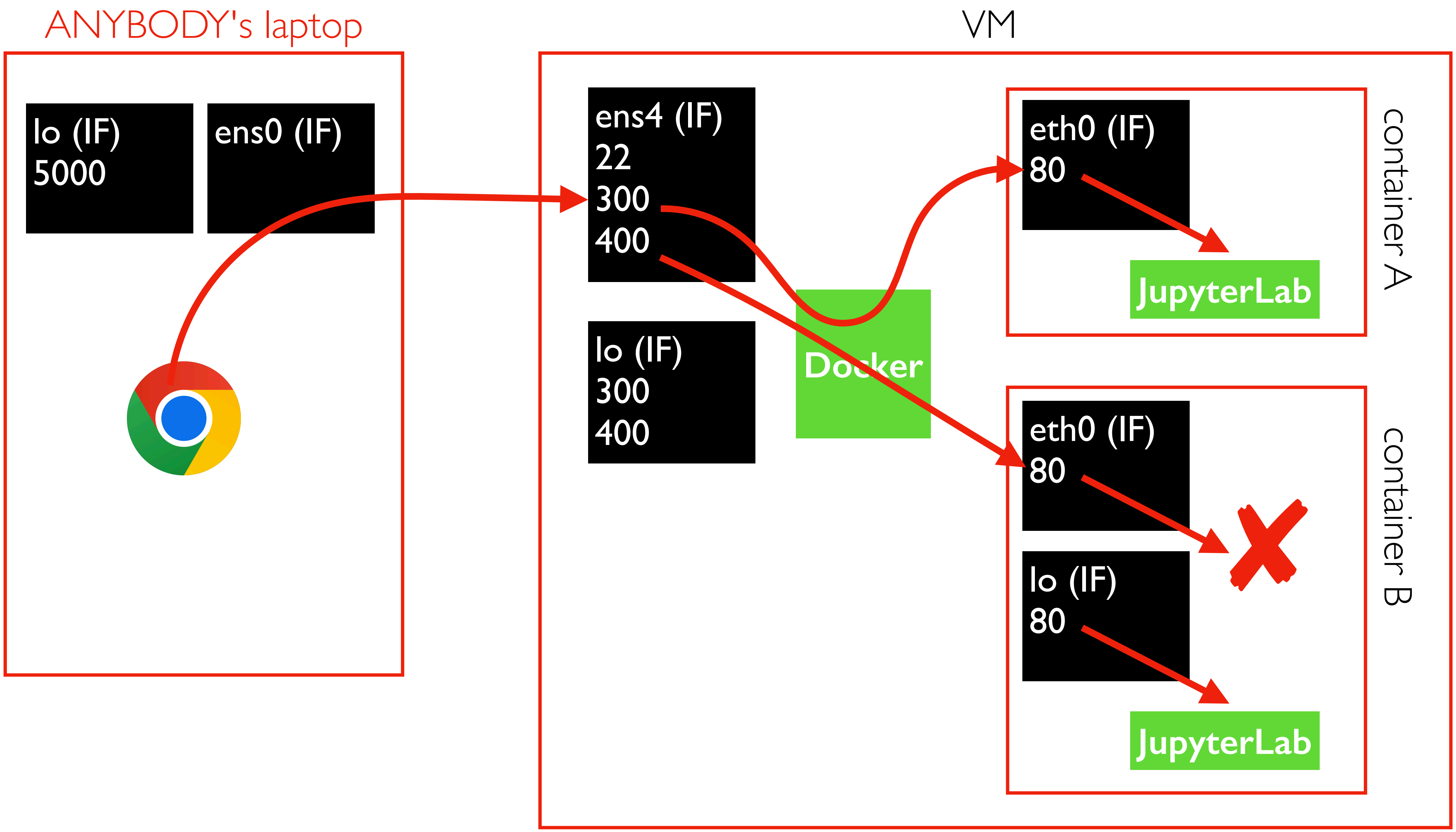
docker run -d **-p** 0.0.0.0:300:80 myimg

Careful, default is to listen on all NICs!

Other security options:

- firewall (block port 300)
- password (in JupyterLab)

Interfaces (IF) and Ports



Port forwarding never goes to loopback inside container

- don't use localhost or 127.0.0.1 inside container!
- easiest: use 0.0.0.0 inside container (for all) to port-forwarded traffic

Demos…

Outline

Docker Port Forwarding

Docker Compose

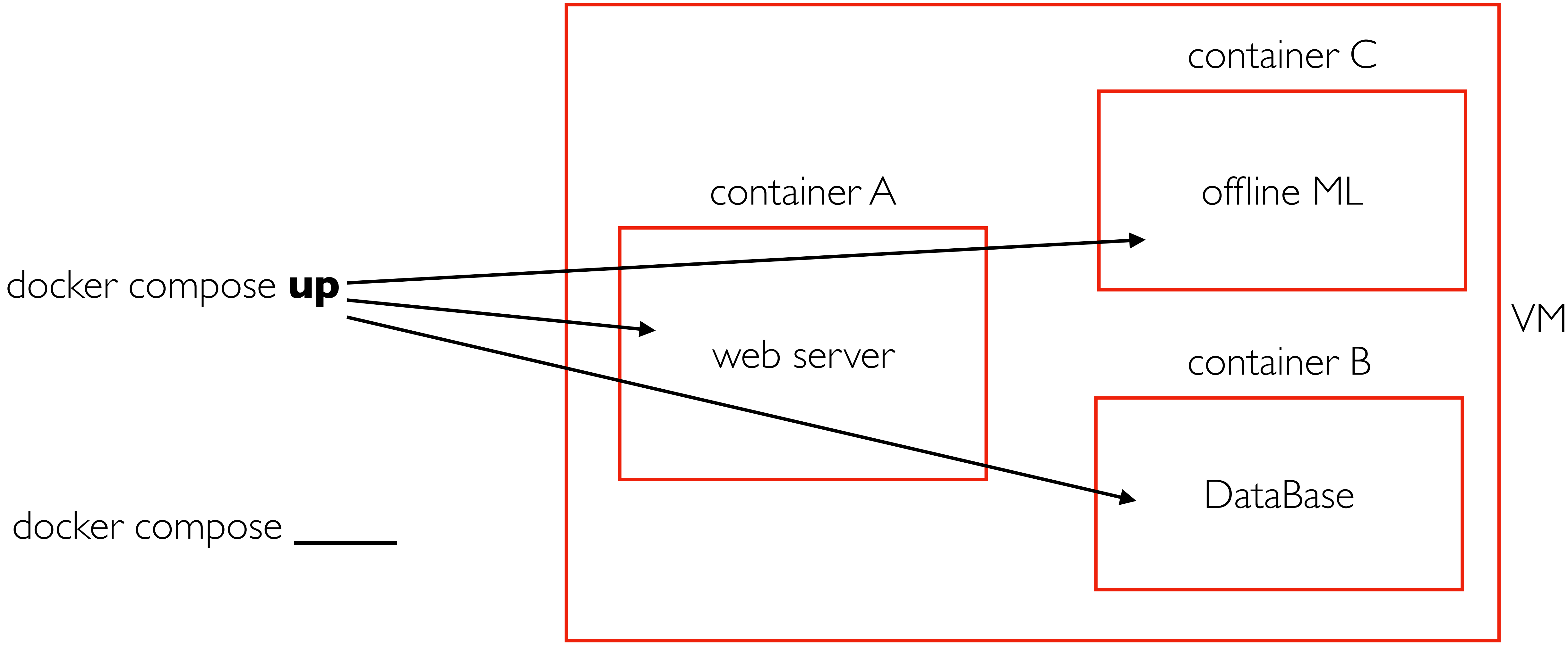
Partitioning and Replication

Container Orchestration

Orchestration lets you deploy many cooperating containers across a cluster of Docker workers.

Kubernetes (K8s) is the most well known.

Docker compose is a simpler tool that lets you deploy cooperating containers to a single worker.



Demos...

Outline

Docker Port Forwarding

Docker Compose

Partitioning and Replication

Data Placement

Say we have large dataset, and many machines.

- can we breakup the dataset (partitioning) so different machines can each help with part of it?
- should we have multiple copies (replication) of the same data so that we don't lose information if a machine fails?

Partitioning

Scenario: we have two computers, and want an app that lets instructors lookup student IDs by name.

computer 1

dataset

Name	Student ID
Aarav Patel	9031231234
Chen Wei	8123456789
Fatima Al-Farsi	7234567890
Hiroshi Tanaka	6345678901
Isabella Rossi	5456789012
John Smith	4567890123
Liam O’Connor	3678901234
Maria Garcia	2789012345
Nia Kofi	1890123456
Yuki Nakamura	1001234567

computer 2

Simple Partitioning

Scenario: we have two computers, and want an app that lets instructors lookup student IDs by name.

dataset

Name	Student ID
Aarav Patel	9031231234
Chen Wei	8123456789
Fatima Al-Farsi	7234567890
Hiroshi Tanaka	6345678901
Isabella Rossi	5456789012
John Smith	4567890123
Liam O'Connor	3678901234
Maria Garcia	2789012345
Nia Kofi	1890123456
Yuki Nakamura	1001234567

computer 1

first half

Name	Student ID
Aarav Patel	9031231234
Chen Wei	8123456789
Fatima Al-Farsi	7234567890
Hiroshi Tanaka	6345678901
Isabella Rossi	5456789012

computer 2

second half

Name	Student ID
John Smith	4567890123
Liam O'Connor	3678901234
Maria Garcia	2789012345
Nia Kofi	1890123456
Yuki Nakamura	1001234567

Challenge: might not easily know which computer to "ask" for a given name (less efficient to ask both each time)

Range Partitioning

If we partition by range, we definitely know which compute to ask for a given name.

dataset

Name	Student ID
Aarav Patel	9031231234
Chen Wei	8123456789
Fatima Al-Farsi	7234567890
Hiroshi Tanaka	6345678901
Isabella Rossi	5456789012
John Smith	4567890123
Liam O'Connor	3678901234
Maria Garcia	2789012345
Nia Kofi	1890123456
Yuki Nakamura	1001234567

computer 1

A-M

Name	Student ID
Aarav Patel	9031231234
Chen Wei	8123456789
Fatima Al-Farsi	7234567890
Hiroshi Tanaka	6345678901
Isabella Rossi	5456789012
John Smith	4567890123
Liam O'Connor	3678901234
Maria Garcia	2789012345

computer 2

N-Z

Name	Student ID
Nia Kofi	1890123456
Yuki Nakamura	1001234567

Challenge: it might be hard to find good split points, especially if the dataset is changing.

Background: Hash Functions

Function: anything => int. Same input, same output. Slightly different input, (usually) very different output.

- hash(“hello world”) = -6807039756899905294
- hash(“hello world”) = -6807039756899905294
- hash("hello word”) = 7093742667072637099

Background: Hash Functions

Function: anything => int. Same input, same output. Slightly different input, (usually) very different output.

- hash(“hello world”) = -6807039756899905294
- hash(“hello world”) = -6807039756899905294
- hash("hello word”) = 7093742667072637099

Observation: MY_NUMBER % N will give produce [0, N-1]

Background: Hash Functions

Function: anything => int. Same input, same output. Slightly different input, (usually) very different output.

- hash(“hello world”) % 10 = 6
- hash(“hello world”) % 10 = 6
- hash("hello word”) % 10 = 9

Observation: MY_NUMBER % N will give produce [0, N-1]

Strategy: if there are N places you could store some data, identify a key within the data, and use hash+modulo to determine where it goes. Lookup by key can find it quickly.

Hash Partitioning

First, choose key column, then hash it, and mod by 2 (to determine which of the 2 computers). If you have a key, can can do the same calculation to find where the whole row lives.

dataset

key

Name	Student ID	hash(Name)
Aarav Patel	9031231234	360993
Chen Wei	8123456789	70525
Fatima Al-	7234567890	913591
Hiroshi	6345678901	121696
Isabella Rossi	5456789012	258452
John Smith	4567890123	438815
Liam	3678901234	588279
Maria Garcia	2789012345	388236
Nia Kofi	1890123456	679776
Yuki	1001234567	160849

computer 1

even hash

Name	Student ID	hash(Name)
Hiroshi	6345678901	121696
Isabella	5456789012	258452
Maria	2789012345	388236
Nia Kofi	1890123456	679776

computer 2

odd hash

Name	Student ID	hash(Name)
Aarav	9031231234	360993
Chen Wei	8123456789	70525
Fatima Al-	7234567890	913591
John	4567890123	438815
Liam	3678901234	588279
Yuki	1001234567	160849

Good: usually balances rows fairly well

Challenge: not good if you want to do lookup for all names in an alphabetic range

Partitioning Vocabulary

TERMINOLOGICAL CONFUSION

What we call a *partition* here is called a *shard* in MongoDB, Elasticsearch, and SolrCloud; it's known as a *region* in HBase, a *tablet* in Bigtable, a *vnode* in Cassandra and Riak, and a *vBucket* in Couchbase. However, *partitioning* is the most established term, so we'll stick with that.

Chapter 6. Partitioning

Replication

Scenario: we have two computers, and want our app to work even if one is down

dataset

Name	Student ID
Aarav Patel	9031231234
Chen Wei	8123456789
Fatima Al-Farsi	7234567890
Hiroshi Tanaka	6345678901
Isabella Rossi	5456789012
John Smith	4567890123
Liam O'Connor	3678901234
Maria Garcia	2789012345
Nia Kofi	1890123456
Yuki Nakamura	1001234567

computer 1

first copy

Name	Student ID
Aarav Patel	9031231234
Chen Wei	8123456789
Fatima Al-Farsi	7234567890
Hiroshi Tanaka	6345678901
Isabella Rossi	5456789012

computer 2

second copy

Name	Student ID
Aarav Patel	9031231234
Chen Wei	8123456789
Fatima Al-Farsi	7234567890
Hiroshi Tanaka	6345678901
Isabella Rossi	5456789012

Challenge: might not easily know which computer to "ask" for a given name (less efficient to ask both each time)

Partitioning and Replication Together

Good: dataset can be bigger than one computer's capacity
Good: if any one computer dies, we don't lose data/availability

computer 1

even hash (copy 1)

Name	Student ID	hash(Name)
Hiroshi	6345678901	121696
Isabella	5456789012	258452
Maria	2789012345	388236
Nia Kofi	1890123456	679776

computer 3

even hash (copy 2)

Name	Student ID	hash(Name)
Hiroshi	6345678901	121696
Isabella	5456789012	258452
Maria	2789012345	388236
Nia Kofi	1890123456	679776

computer 2

odd hash (copy 1)

Name	Student ID	hash(Name)
Aarav	9031231234	360993
Chen Wei	8123456789	70525
Fatima Al-	7234567890	913591
John	4567890123	438815
Liam	3678901234	588279
Yuki	1001234567	160849

computer 4

odd hash (copy 2)

Name	Student ID	hash(Name)
Aarav	9031231234	360993
Chen Wei	8123456789	70525
Fatima Al-	7234567890	913591
John	4567890123	438815
Liam	3678901234	588279
Yuki	1001234567	160849