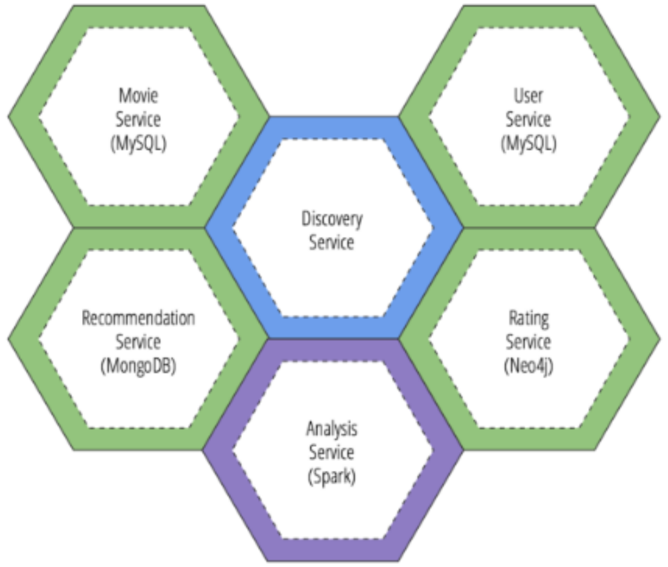
UNIVERSITÀ DEGLI STUDI DI UDINE

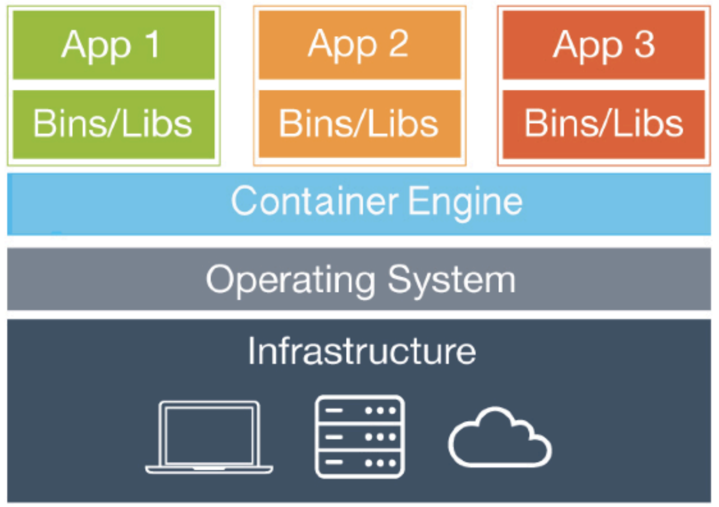
Bigraphical models for Container-based Systems

Marino Miculan, Fabio Burco
MADS lab — DMIF, University of Udine
marino.miculan@uniud.it

Microservice-oriented architectures and containers

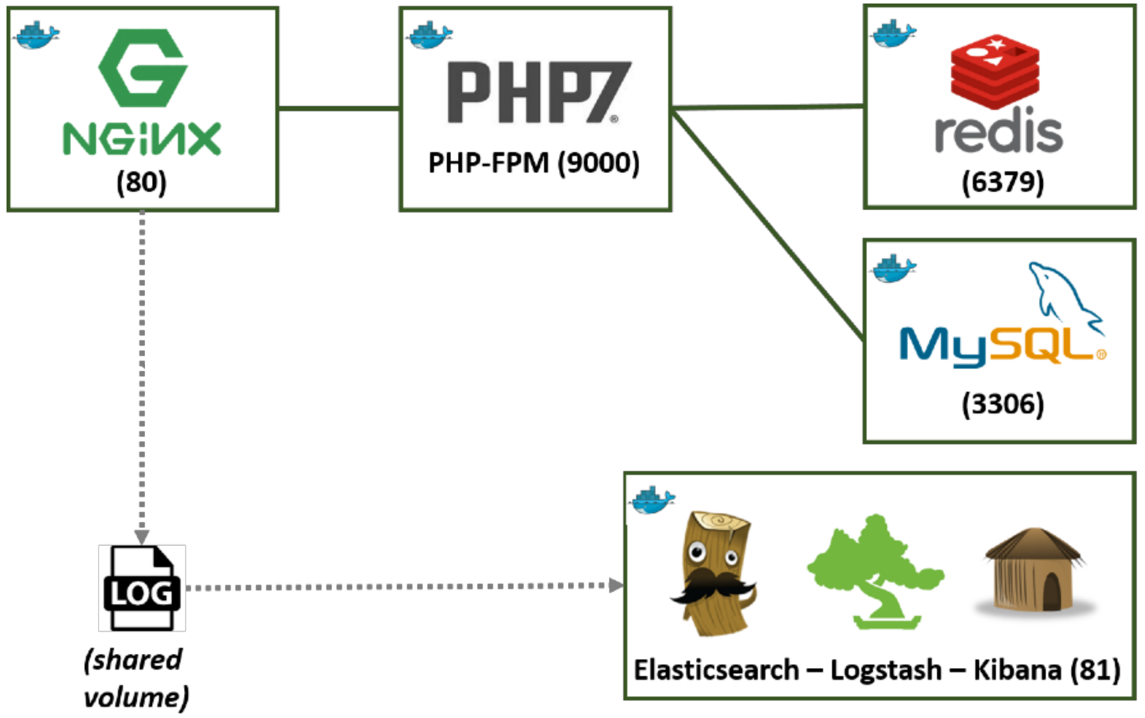
- **Microservice-oriented architecture**
  - Flexible, Scalable, supporting dynamic deployment and reconfiguration, etc.
- **Containers** are emerging as a good way for implementing Microservices
  - Ensure execution separation, separation of tasks, portability
  - Lighter than virtual machines
  - Support service and component **composition**





Containers are made to be composed

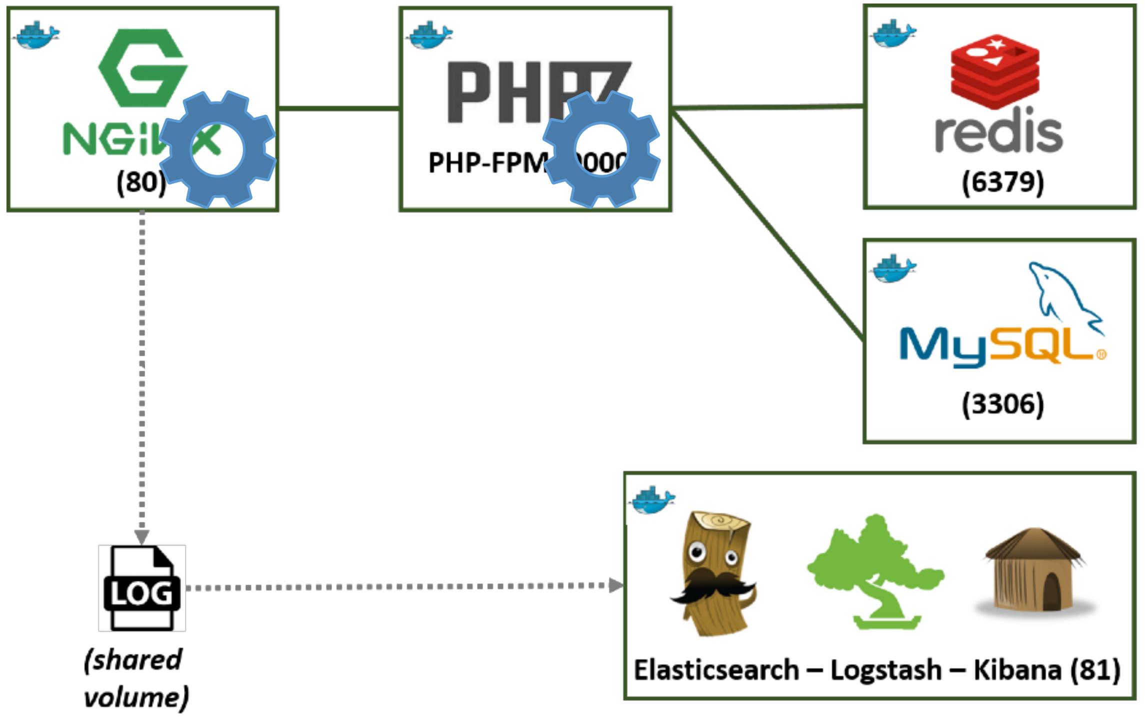
- Containers can be composed to form larger systems
- Two different compositions:
  - **Horizontal***: containers are on a par, and communicate through channels, volumes, networks



Containers are made to be composed

- Containers can be composed to form larger systems
- Two different compositions:
  - **Horizontal***: containers are on a par, and communicate through channels, volumes, networks
  - **Vertical***: containers may have "holes", to be filled with application specific code, processes… (by developers or at deployment)

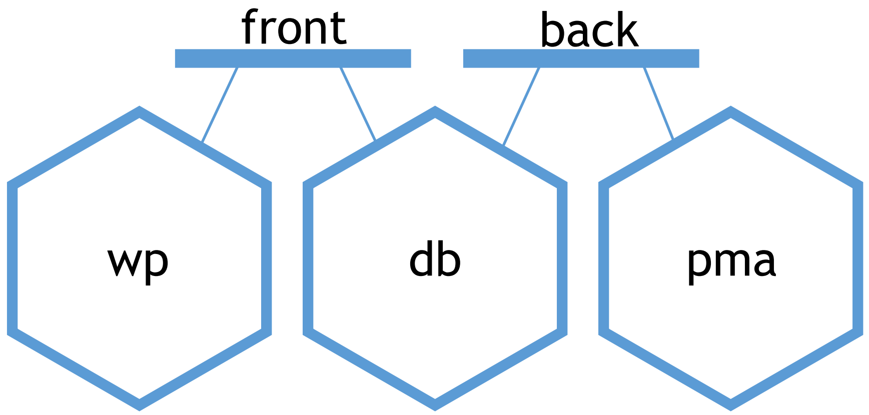
* = my naming, not official



Containers are made to be composed

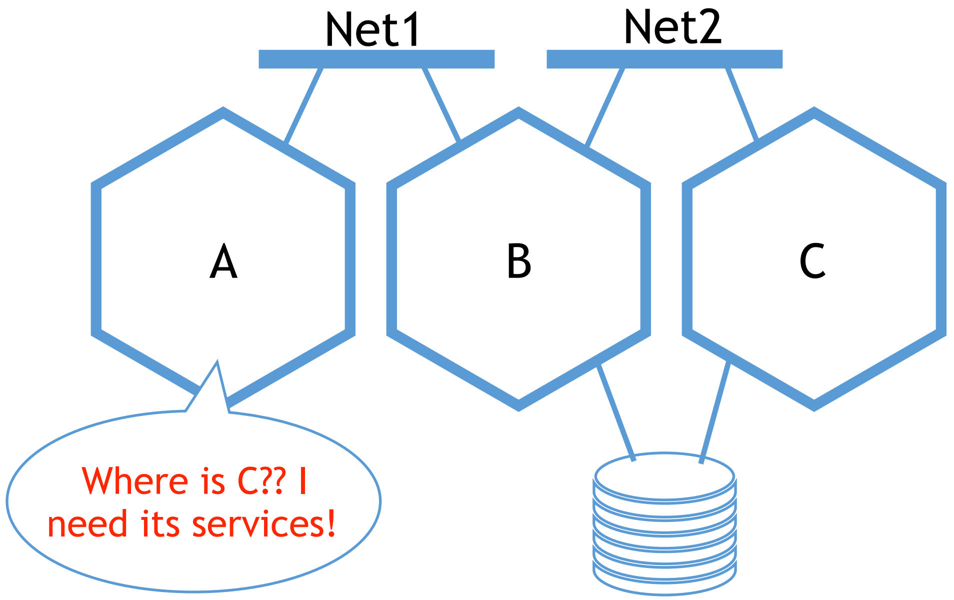
- Composition is defined by YAML files declaring
  - Networks
  - Volumes (possibly shared)
  - For each container
    - Name
    - Images
    - Networks which are connected to
    - Port remappings
    - Links to other services
- The configuration is then fed to an orchestration tool (`docker-compose`) which creates all the containers, the networks, the connections, etc. and launches the system

```
services:
  wp:
    image: wordpress
    links:
      - db
    ports:
      - "8080:80"
    networks:
      - front
    volumes:
      - datavolume:/var/www/data:ro
  db:
    image: mariadb
    expose:
      - "3306"
    networks:
      - front
      - back
  pma:
    image: phpmyadmin/phpmyadmin
    links:
      - db:mysql
    ports:
      - "8181:80"
    volumes:
      - datavolume:/data
    networks:
      - back
networks:
  front:
    driver: bridge
  back:
    driver: bridge
volumes:
  datavolume:
    external: true
```



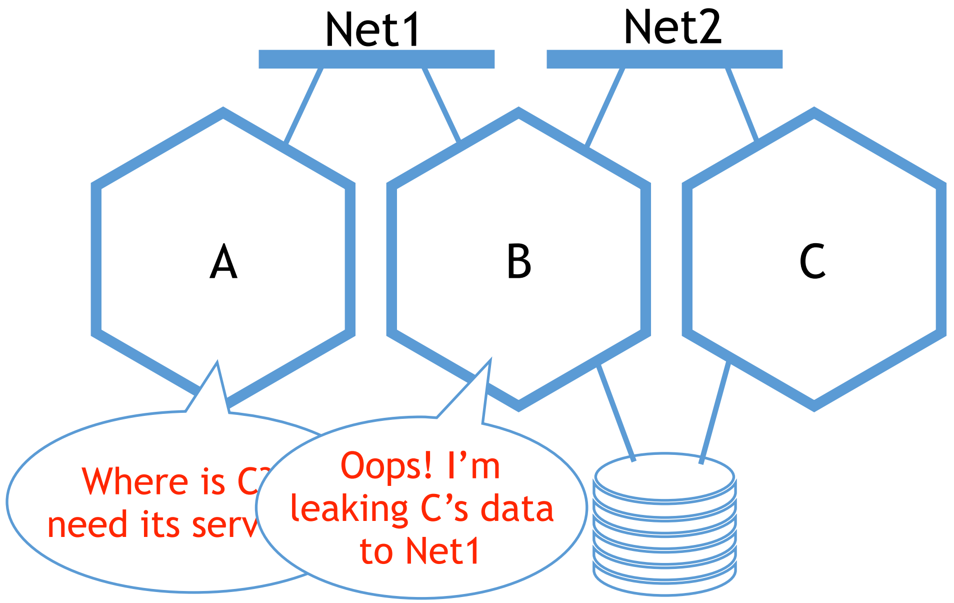
How to check if composition is *correct*?

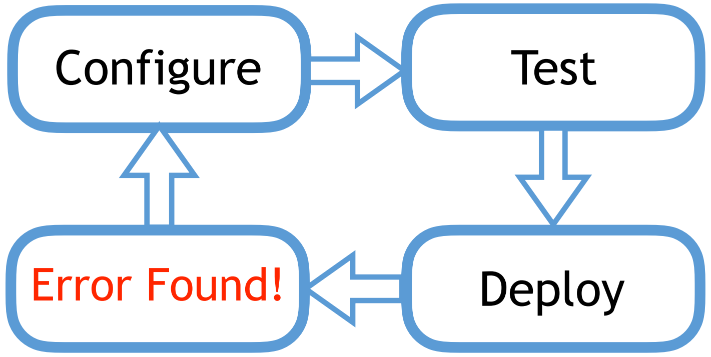
- A misconfiguration may lead to issues during composition, or (worse) at runtime. E.g.:
  - A container requests access to a missing services, or to a service which is not connected to by a network



How to check if composition is *correct*?

- A misconfiguration may lead to issues during composition, or (worse) at runtime. E.g.:
    - A container requests access to a missing services, or to a service which is not connected to by a network
    - Ambiguous declaration of services
    - Security policies violations, e.g. sharing networks or volumes which should not be
- Dynamic reconfiguration can break properties
- Composition tools check only very basic aspects
- Common approach: *try-and-error*
    - Expensive and not safe enough


Net1
Net2
A
B
C
Where is C? I need its serv
Oops! I'm leaking C's data to Net1



How to check if composition is *correct*?

- Hence, we need **tools** for analyzing, verifying (and possibly manipulate) container configurations
    - Before executing the system (static analysis), or at runtime
- For this we need a ***formal model of containers and services composition***
- This model should support:
    - Logical connections of components
    - Horizontal and vertical (nesting) composition of components
    - Dynamic reconfiguration
    - Different granularities
    - Flexibility and openness
    - …

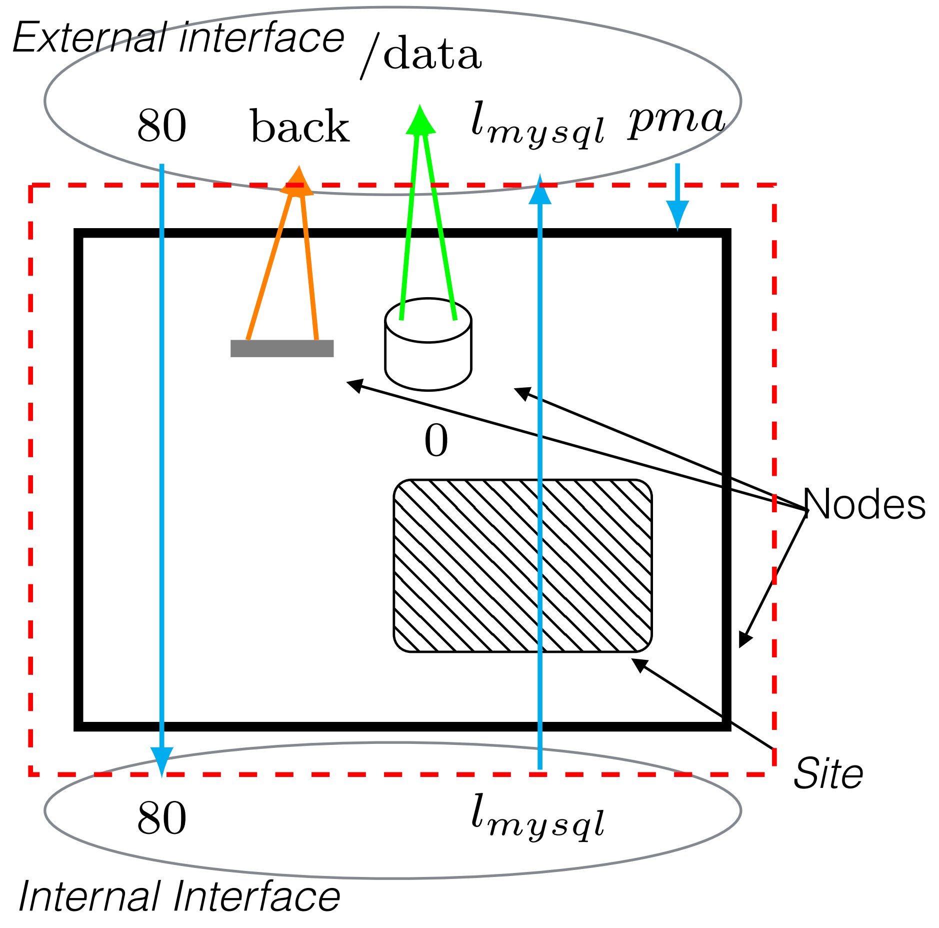
How to check if composition is *correct*?

- Hence, we need **tools** for analyzing, verifying (and possibly manipulate) container configurations
  - Before executing the system (static analysis), or at runtime
- For this we need a ***formal model of containers and services composition***
- This model should support:
  - Logical con
  - Horizontal a   ents
  - Dynamic re
  - Different gr
  - Flexibility a
  - …

Bigraphs (Milner, 2003): a general (meta)model for distributed communicating systems, supporting composition and nesting.

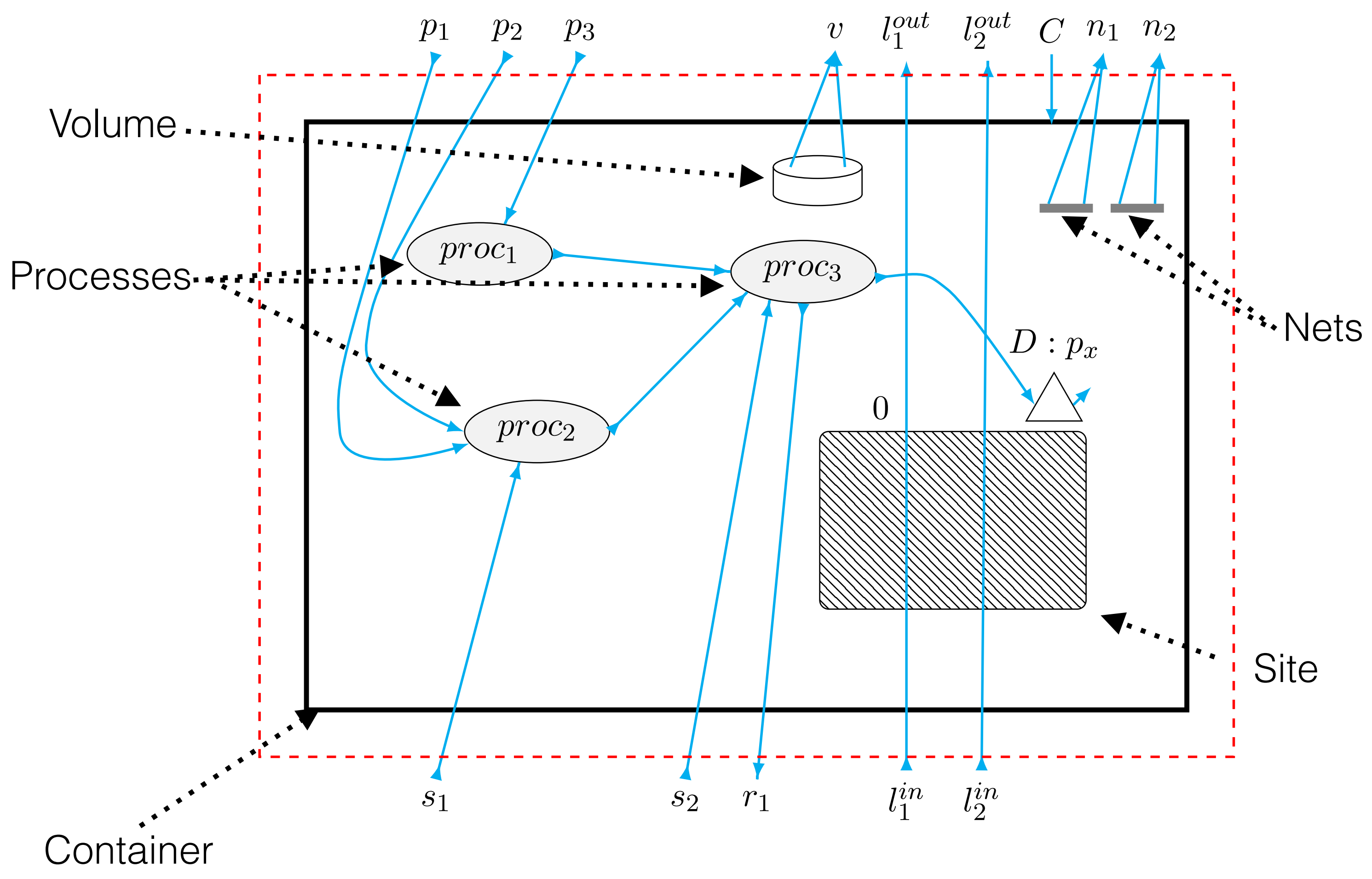
Local Direct Bigraphs

- For containers, we have introduced **local directed bigraphs**: special graphs where
  - Nodes have assigned a type, specifying arity and polarity (represented by different shapes) and can be nested
  - *Sites* represent "holes" which can be filled with other bigraphs
  - Arcs can connect nodes to nodes (respecting polarities) or to names in *internal* and *external interfaces* (with locality)



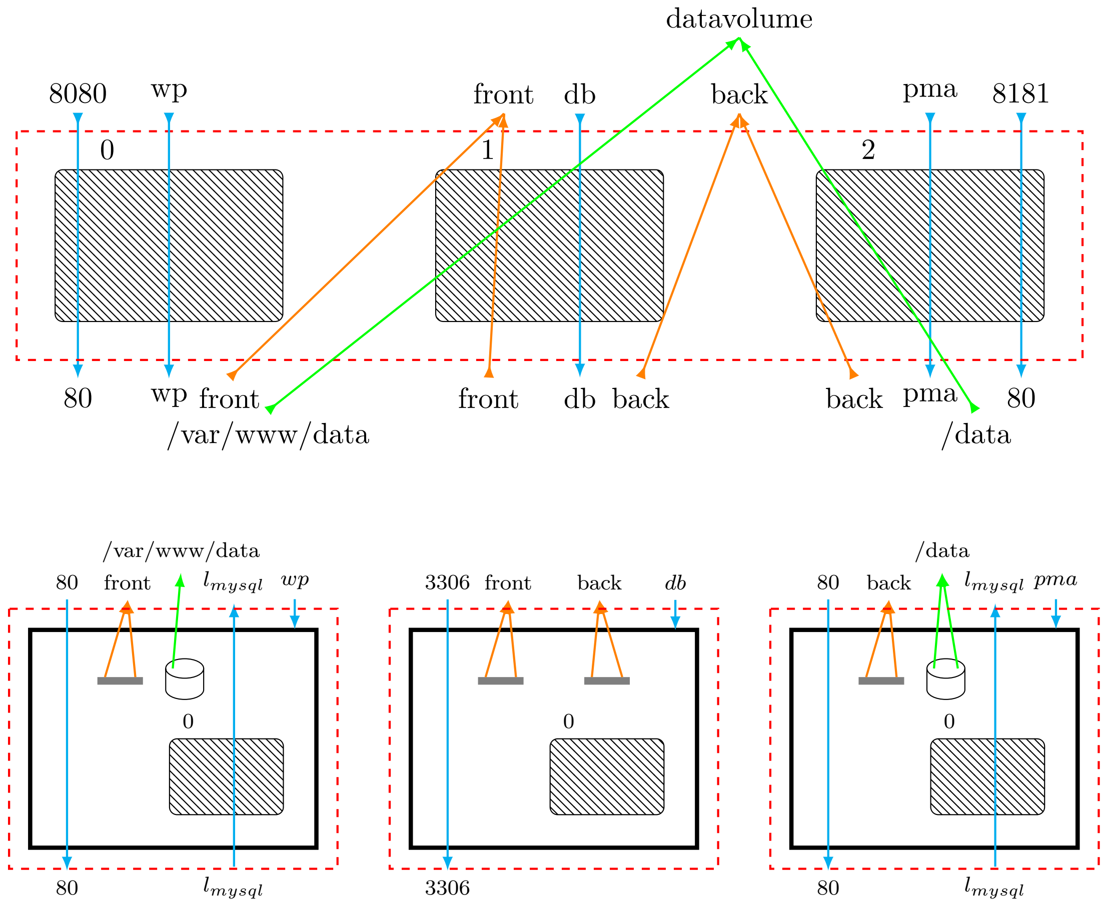
Containers are local directed bigraphs!

- Each container can be represented as a ldb, whose interfaces contain the name of the container, the exposed ports, required volumes and networks, etc.
- Important remark: this is not just a picture. This is the graphical representation of a mathematical object (an arrow of a specific category)



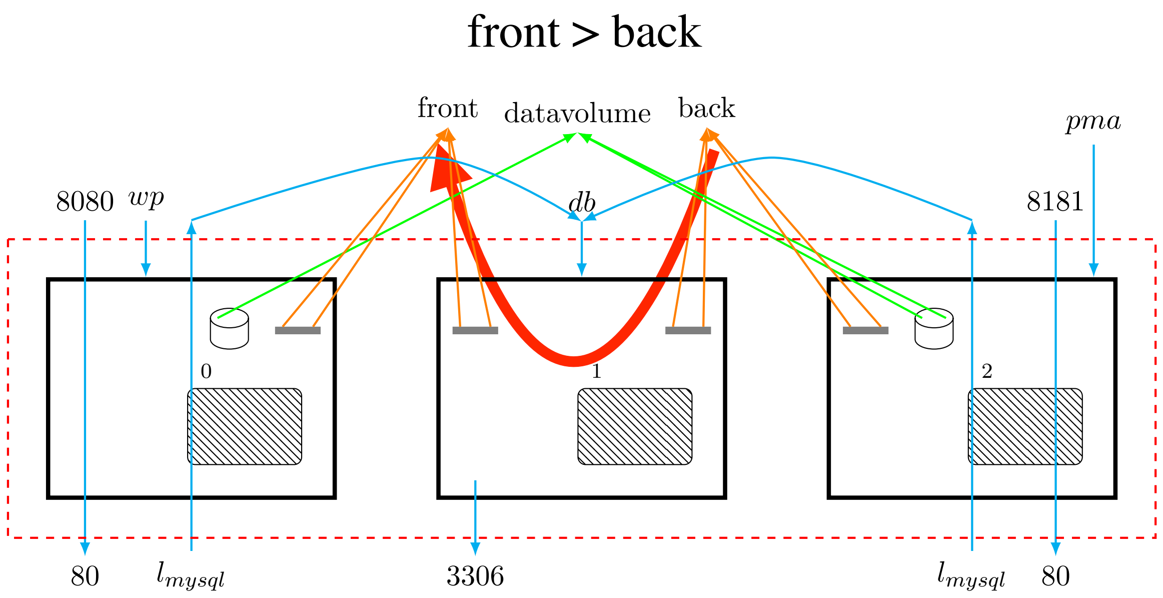
And composition is another bigraph itself!

- Bigraphs (being arrows in a category) compose when their interfaces correspond
- Composition of containers as done by `docker-compose` = composition of corresponding bigraphs inside a "deployment bigraph" specifying volumes, networks, name and port remapping, etc.
- The deployment bigraph can be obtained automatically from the YAML configuration file



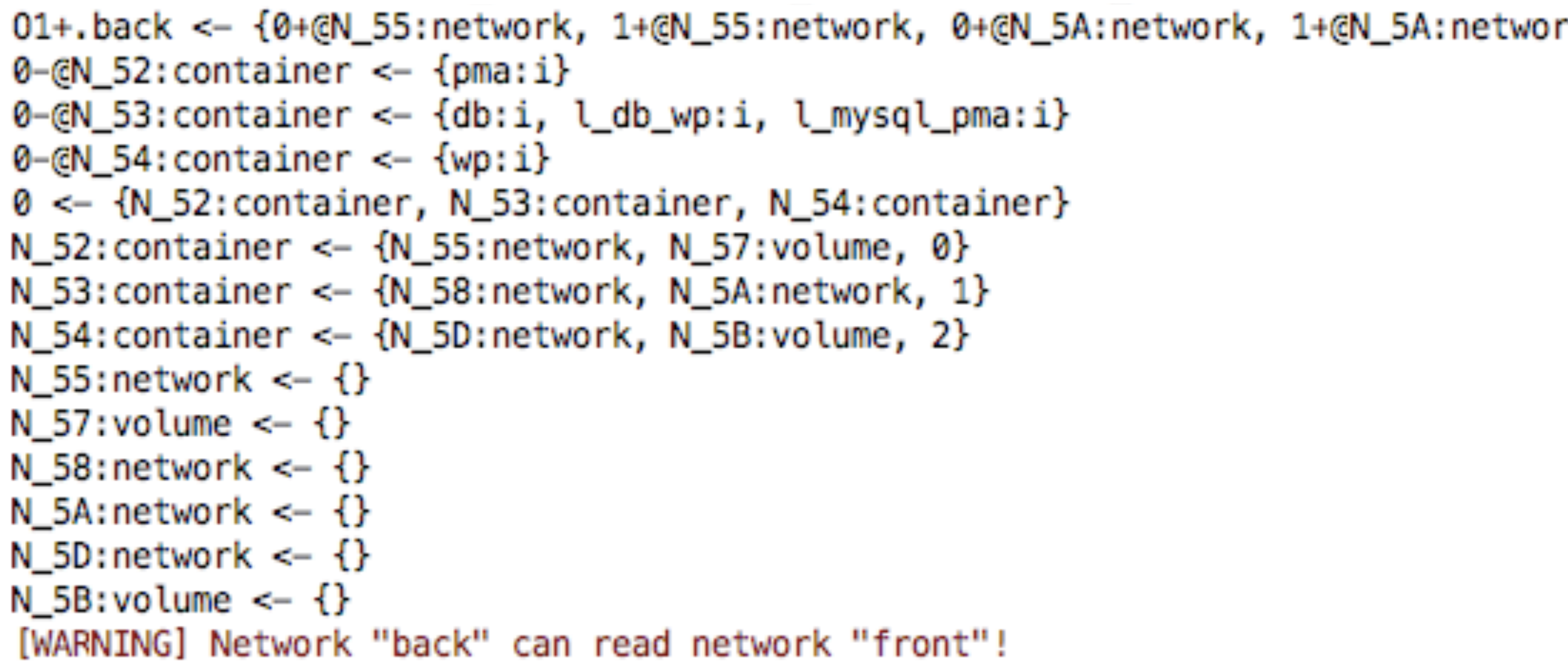
Another example: Safe network separation

- assume that networks (or volumes) have assigned different security levels (e.g "admin > guests > public").
- Security property to guarantee:
    - "data from a higher security network cannot leak into a lower security network, even going through different containers"
- Can be reduced to a *reachability problem* on the corresponding bigraph
    - "For each pair of nets m,n such that m > n, there is no directed path from n to m"
    - If this is the case, then the configuration respects the separation policy. Otherwise, an information leakage is possible



Prototype tool: `docker2ldb`

- `docker2ldb` is a CLI tool (written in Java using the *jLibBig* library) which
  - reads a docker-compose configuration file
  - builds the corresponding deployment bigraph object
  - checks for valid connections and network separation

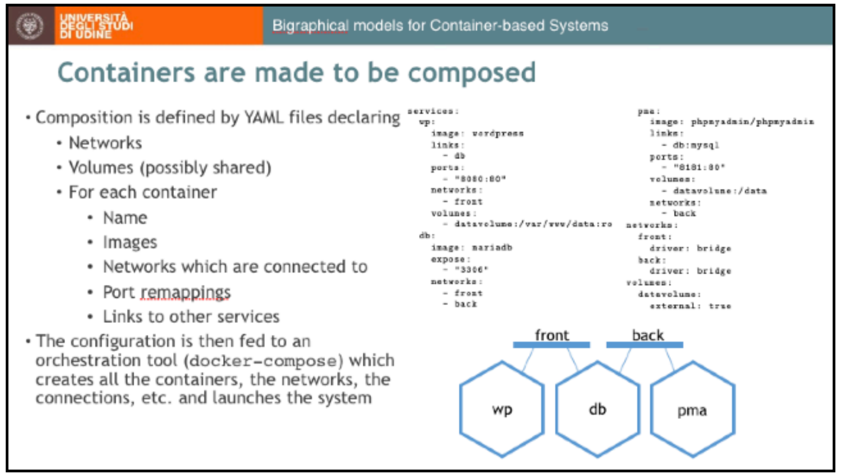


```
01+.back <- {0+@N_55:network, 1+@N_55:network, 0+@N_5A:network, 1+@N_5A:networ
0-@N_52:container <- {pma:i}
0-@N_53:container <- {db:i, l_db_wp:i, l_mysql_pma:i}
0-@N_54:container <- {wp:i}
0 <- {N_52:container, N_53:container, N_54:container}
N_52:container <- {N_55:network, N_57:volume, 0}
N_53:container <- {N_58:network, N_5A:network, 1}
N_54:container <- {N_5D:network, N_5B:volume, 2}
N_55:network <- {}
N_57:volume <- {}
N_58:network <- {}
N_5A:network <- {}
N_5D:network <- {}
N_5B:volume <- {}
[WARNING] Network "back" can read network "front"!
```

Conclusions

- **Done:**
    - Proposed a bigraph-based formal model for containerised systems
    - Captures logical connections of components and processes, nesting of components, composition of containers
    - Applicable for, e.g., static analysis of container systems
    - Implemented prototype checker tool
- **To do:**
    - Formalisation of other static properties
    - Consider **dynamics** - in particular, *system reconfiguration*
        - Dynamics in bigraphs are represented by *graph rewriting rules*
        - Can be used to represent horizontal scalability, container replacement / update, etc.
        - New "temporal" safety invariants could be verified

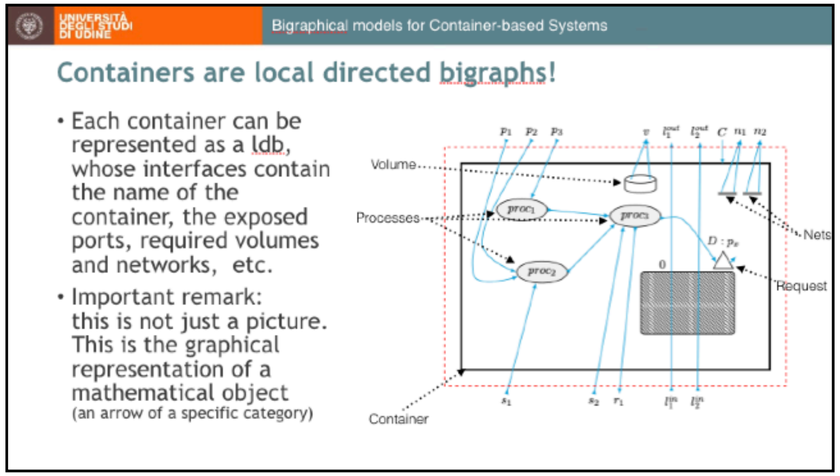
Thanks for your attention! Questions?



Containers are made to be composed

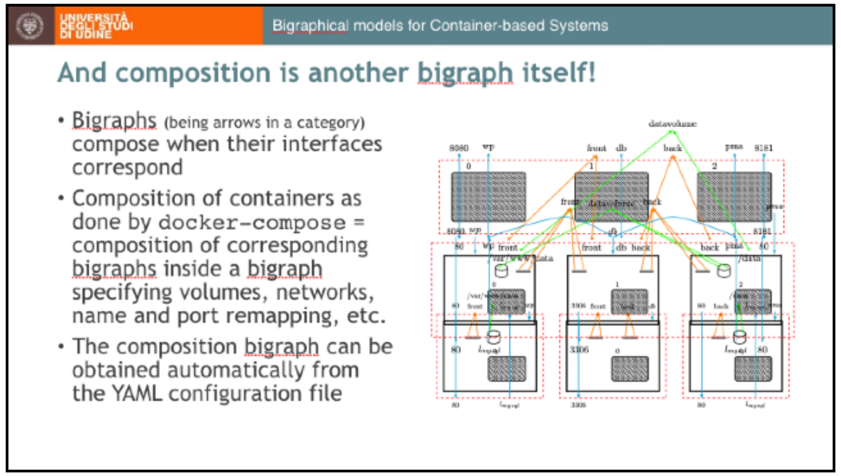
- Composition is defined by YAML files declaring
  - Networks
  - Volumes (possibly shared)
  - For each container
    - Name
    - Images
    - Networks which are connected to
    - Port remappings
    - Links to other services
- The configuration is then fed to an orchestration tool (docker-compose) which creates all the containers, the networks, the connections, etc. and launches the system

```
services:                                   pma:
  wp:                                         image: phpmyadmin/phpmyadmin
    image: wordpress                          links:
    links:                                      - db:mysql
      - db                                    ports:
    ports:                                      - "8181:80"
      - "8080:80"                             volumes:
    networks:                                   - datavolume:/data
      - front                                 networks:
    volumes:                                    - back
      - datavolume:/var/www/data:ro         networks:
  db:                                         front:
    image: mariadb                              driver: bridge
    expose:                                   back:
      - "3306"                                  driver: bridge
    networks:                               volumes:
      - front                                 datavolume:
      - back                                    external: true
```



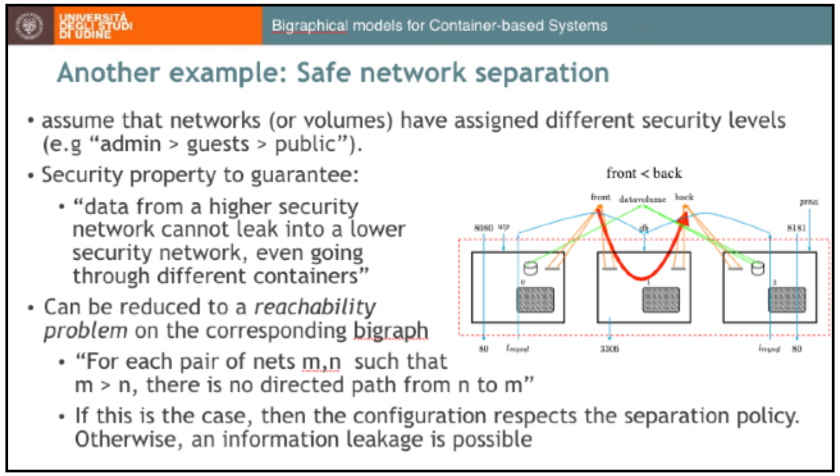
Containers are local directed bigraphs!

- Each container can be represented as a ldb, whose interfaces contain the name of the container, the exposed ports, required volumes and networks, etc.
- Important remark: this is not just a picture. This is the graphical representation of a mathematical object (an arrow of a specific category)



And composition is another bigraph itself!

- Bigraphs (being arrows in a category) compose when their interfaces correspond
- Composition of containers as done by docker-compose = composition of corresponding bigraphs inside a bigraph specifying volumes, networks, name and port remapping, etc.
- The composition bigraph can be obtained automatically from the YAML configuration file



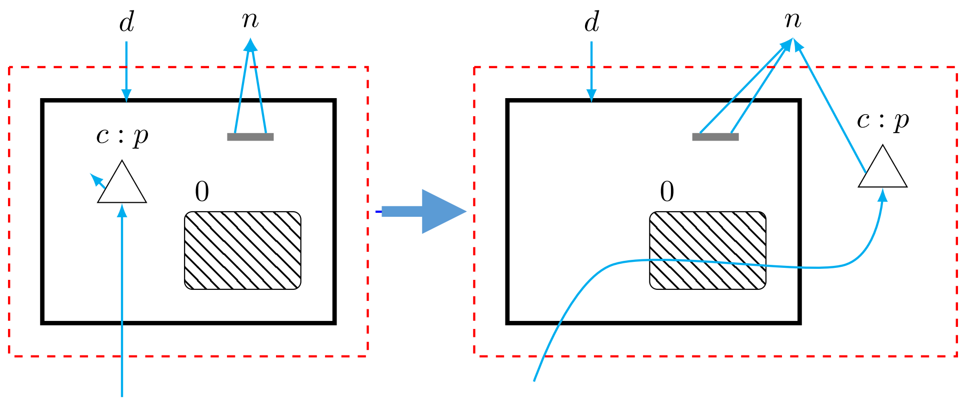
Another example: Safe network separation

- assume that networks (or volumes) have assigned different security levels (e.g "admin > guests > public").
- Security property to guarantee:
  - "data from a higher security network cannot leak into a lower security network, even going through different containers"
- Can be reduced to a *reachability problem* on the corresponding bigraph
  - "For each pair of nets m,n such that m > n, there is no directed path from n to m"
  - If this is the case, then the configuration respects the separation policy. Otherwise, an information leakage is possible

marino.miculan@uniud.it

Dynamics is defined by graph rewriting rules

- A graph rewriting rule is a pair of bigraphs (*redex-reactum*).
- Example:



- A rule can be applied to a bigraph when the left-hand side matches a part of the bigraph
  - Then the matching part is replaced by the reactum
- A rule can replace / move components, change connections, etc…
- A Bigraphic Reactive System is defined by a set of rules