Distributed applications: A challenge for systems, networks, and application development

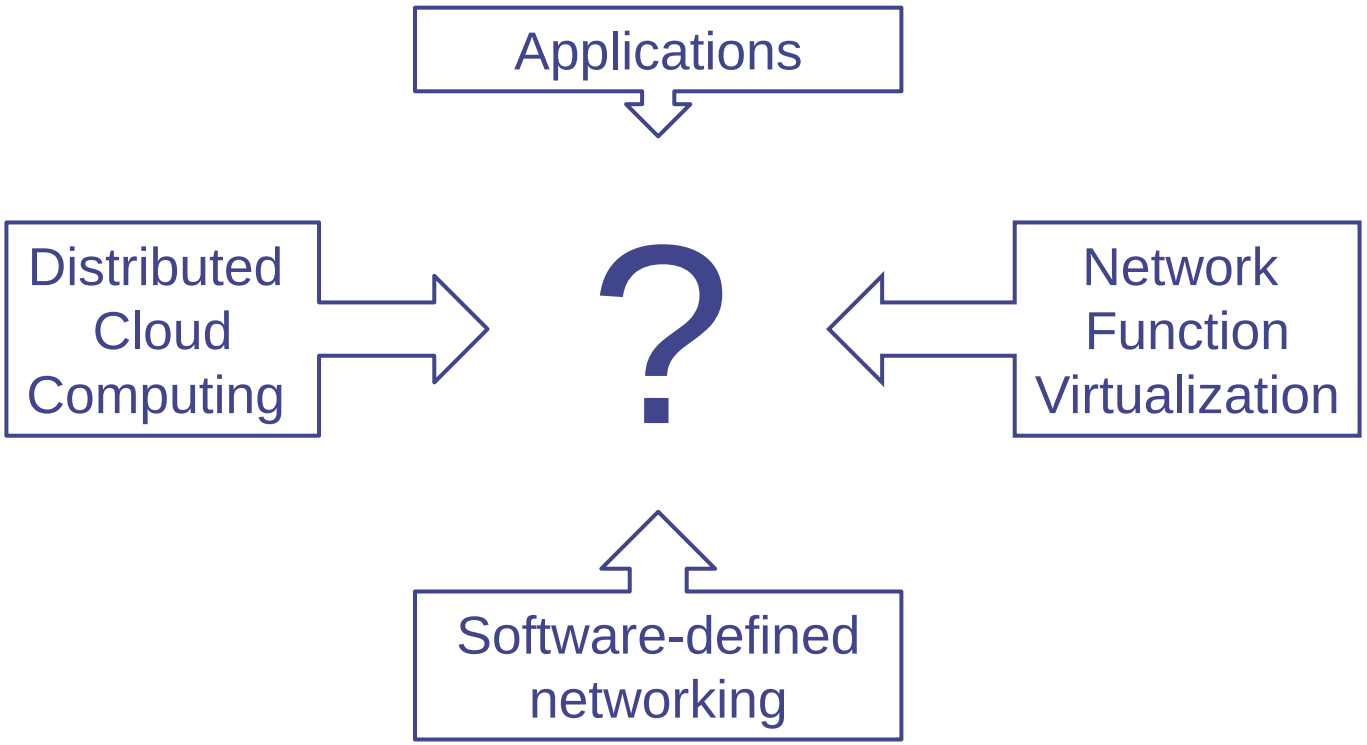
Holger Karl
Paderborn University

Computer Networks Group
Universität Paderborn

Claims

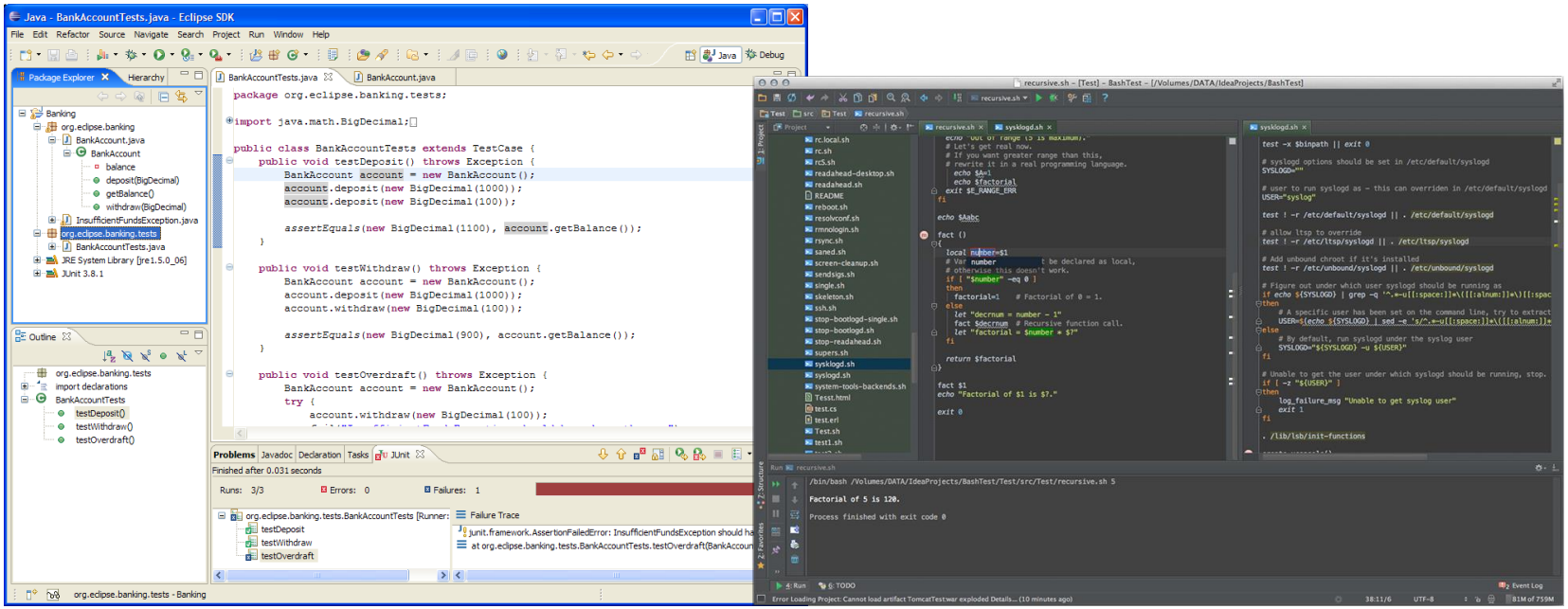
- Developing and deploying applications: Together
  - Buzzword: Continuous software engineering, CI/CD
- Deploying: Applications and infrastructure
  - Buzzword: Infrastructure as code
- Education issues

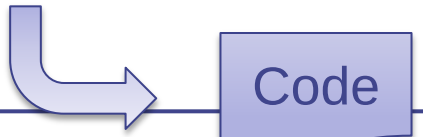
Key ingredients



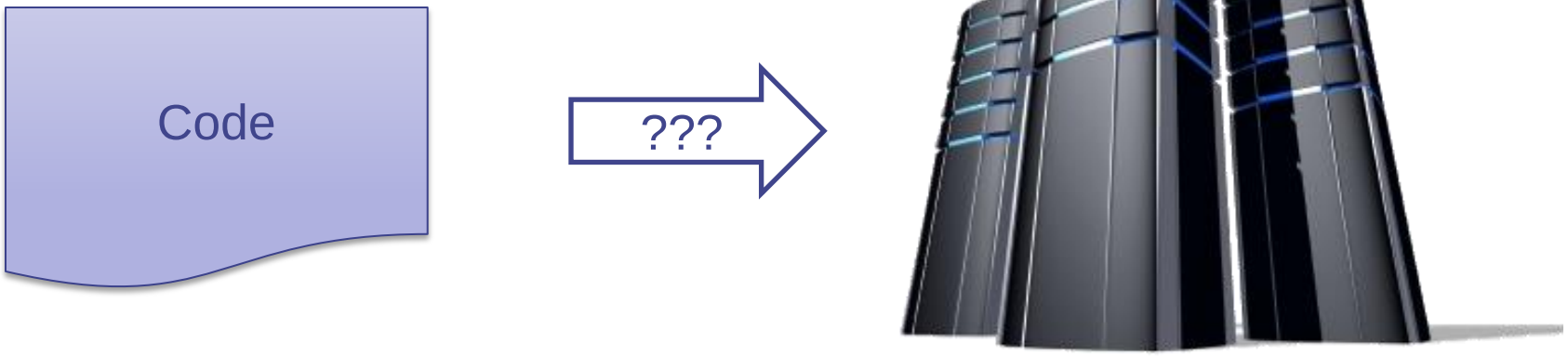
Developing old school

- Monolithic IDEs producing monolithic code





Deploying?



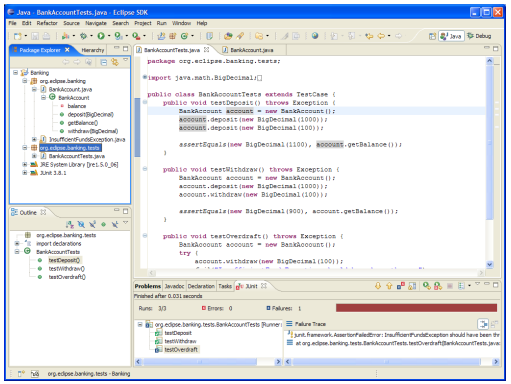
Deploying, old school

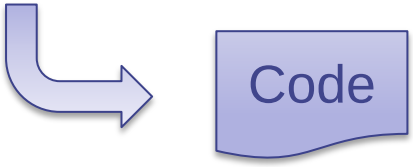
```
$ # download
$ wget http://www.developer.com/app.tgz
$ tar xfz app.tgz
$ # compile and build
$ cd app
$ ~/app> ./configure
$ ~/app>  make
$ ~/app>  make install
$ # run
$ ~/app> ./app
```

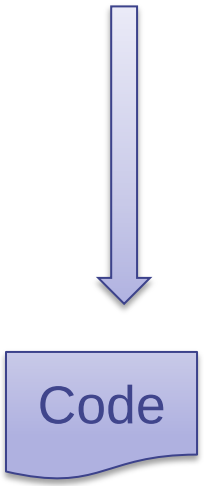
- Great for single application, single machine

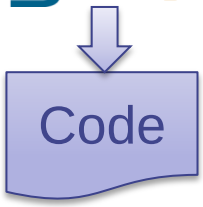
Developing, ca. 2017

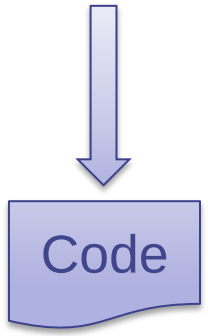
- Own code, plus tons of modules, libraries, **microservices**, …











Deploying using virtual machines

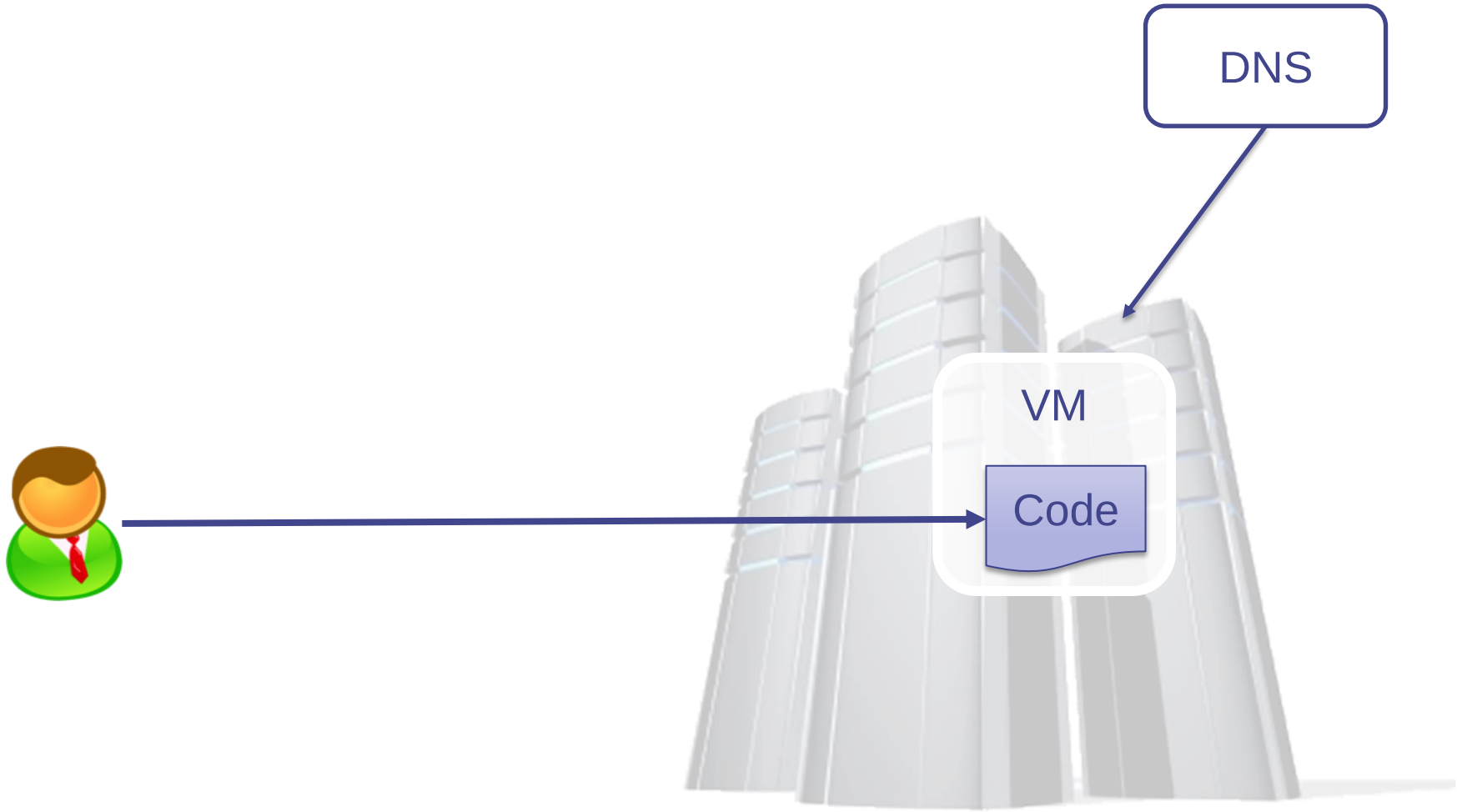
```
$ # download description
$ wget http://www.developer.com/app.tgz
$ tar xfz app.tgz
$ cd app
$ # edit configuration:
$ vi ansible-playbook.yml
$ # download actual image, configure it
$ vagrant --provision up
$ # tell DNS where to find the app:
$ curl --request POST www.mydns.com -d {'fdqn': myapp.com, 'ip':
1.2.3.4}
```

Customize a ready-made image to local needs

E.g., network addresses

- Great for simple applications
- Even inside, e.g., VMWare clusters

Setup after deployment



Deploying complex applications – Manually

- Anything real will need multiple machines

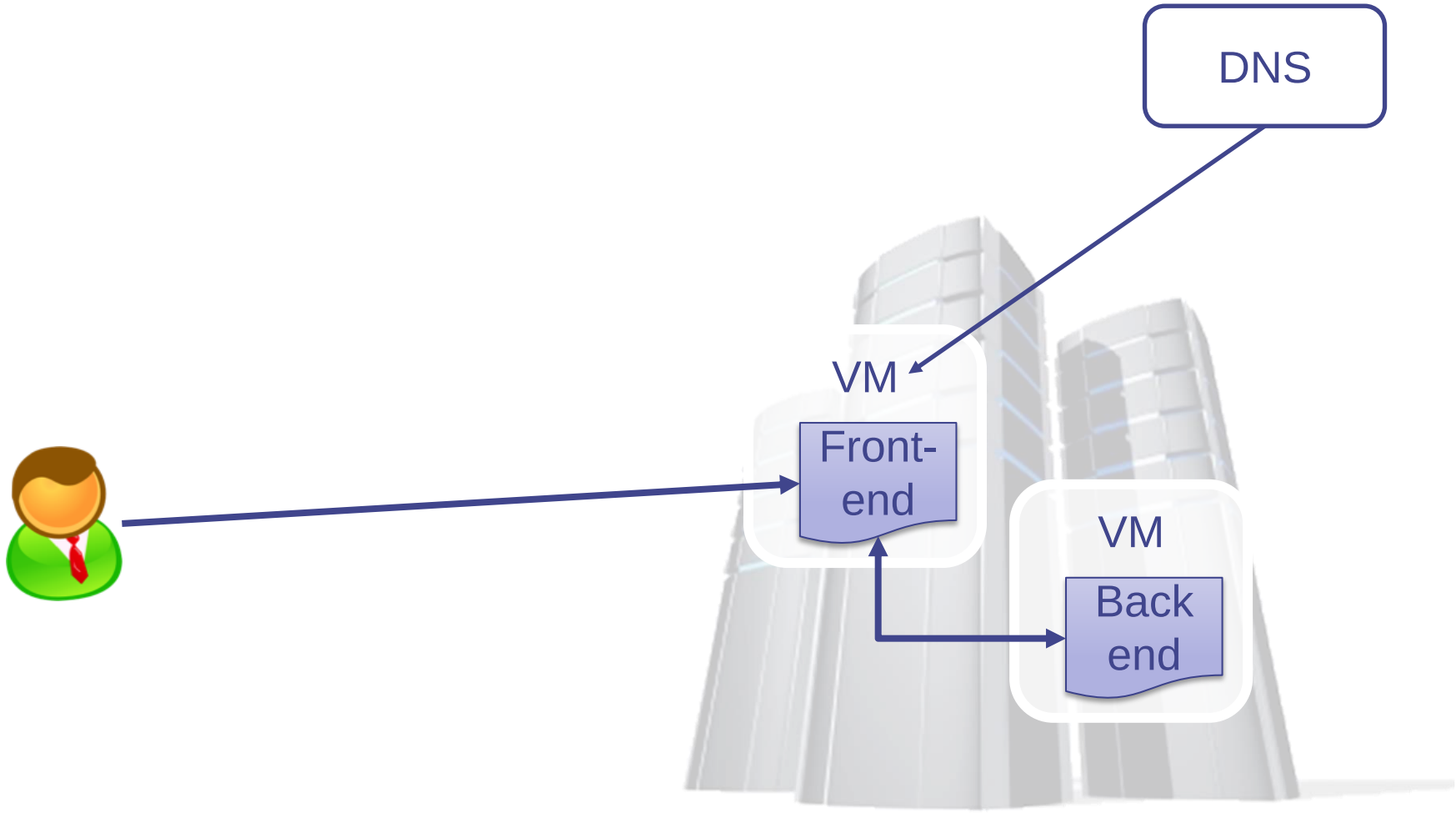
```
$ wget http://www.developer.com/app.tgz
$ tar xfz app.tgz
$ # a frontend server:
$ cd app/frontend
$ # complex editing of network configuration
$ app/frontend> vi ansible-playbook.yml
$ app/frontend> vagrant --provision up
$
$ # a backend server:
$ cd ../app/backend
$ # complex editing of network configuration
$ app/backend> vi ansible-playbook.yml
$ app/backend> vagrant --provision up
$ # tell DNS where to find the app:
$ curl --request POST www.mydns.com -d {'fdqn': myapp.com, 'ip':
frontend_ip}
```

Tell frontend how to talk to backend: IP, ports, keys, …

Tell backend which frontend to accept: IP, ports, keys, …

- Getting slightly less great

Setup after deployment



Scaling out?

- Load increases:
  need another frontend

Start second frontend

Integrate it with backend

```
$ # copy frontend configuration:
$ cp -r app/frontend app/frontend2
$ cd app/frontend2
$ app/frontend2> # complex editing of network configuration
$ app/frontend> vi ansible-playbook.yml
$ # bring up a new fronend instance
$ app/frontend> vagrant --provision up
$
$ # tell backend to accept new frontend
$ cd ../app/backend
$ aap/backend> vagrant ssh
[backend] $ vi backend.config                # add the second frontend
server
[backend] $./backend restart                 # trigger reloading of
config
```

Scaling out?

- But wait, there's more:

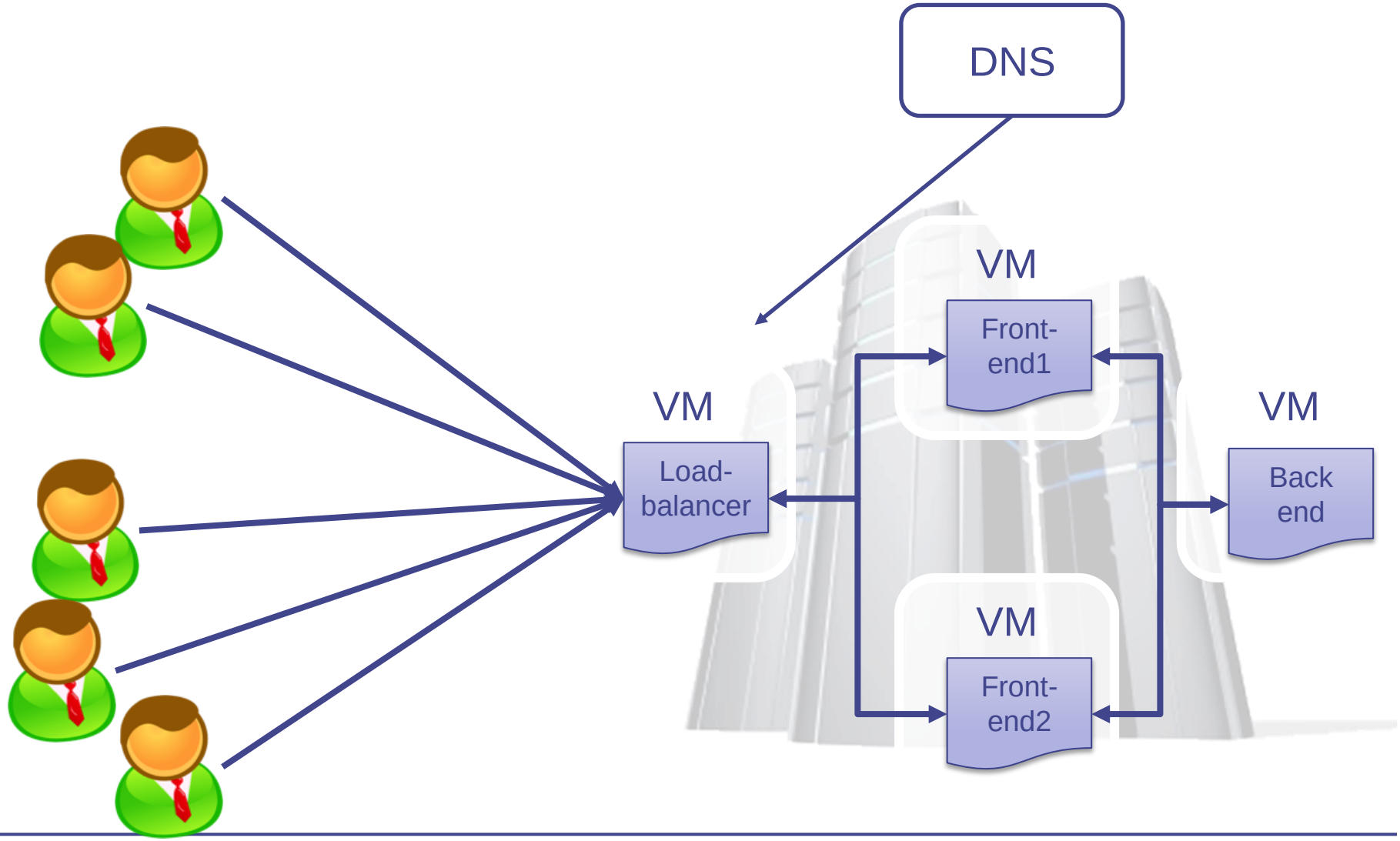
```
$ wget http://www.loadbalancer.com/lb.tgz
$ tar xfz lb.tgz
$ cd lb
$ # complex editing of lb configuration
$ lb> vagrant --provision up
$ cd ..
$ ssh firewall                          # reconfigure          firewall
$ # tell DNS where to find the app:
$ curl --request POST www.mydns.com -d {'fdqn': myapp.com, 'ip':
lb_ip}
```

Distribute traffic to two frontends: Loadbalancer needed

Firewall needs to be told to allow traffic

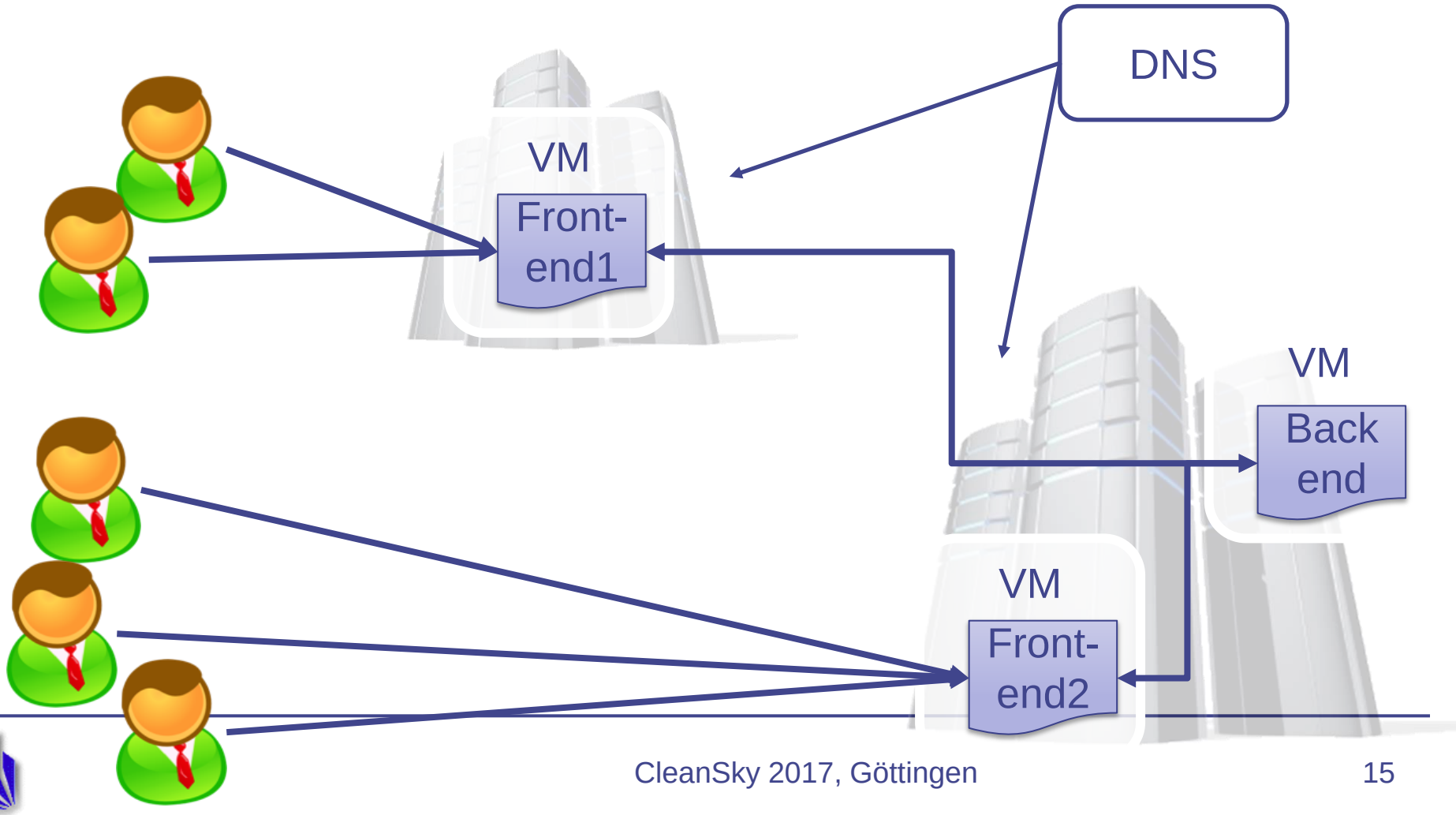
DNS should point to loadbalancer

Setup after deployment



Frontends at different sites?

- Wouldn't this make more sense?



Deploying frontends at remote sites

```
$ # configure frontend2 to run elsewhere
$ cd app/frontend2
$ vagrant halt
$ # put IP of other site into configuration
$ vagrant up
$
$ # drop loadbalancer
$ # reconfigure firewall at both sites
$ # reconfigure DNS
$ # reconfigure backend
```

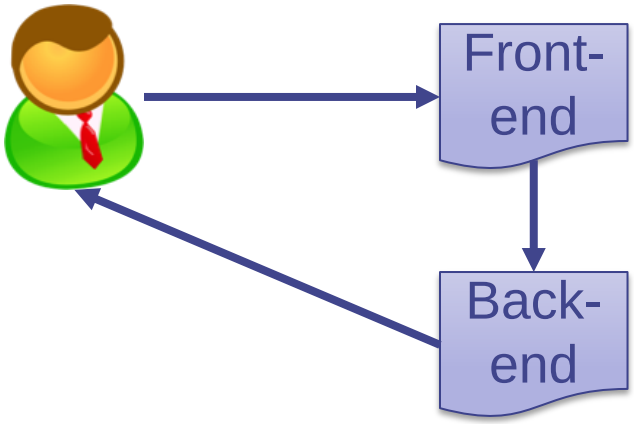
- Definitely not great!

What do we need?

- Get rid of manual configuration via console

- **Describing** configuration of a composed application
  - Its actual application-oriented components
  - Support components like firewalls, …
  - Scaling properties
- **Deploying** such a composed application
- **Configuring** underlying infrastructure

Issues



Describing complex applications – Templates

- **Templates** describe components in an application
- Needs
  - Which components
  - How interconnected
  - How to scale
  - Where to place

Template examples

- OpenStack: HEAT Orchestration Template (HOT)
  - HEAT: OpenStack's orchestration plugin
  - Composition, interconnected, limited scaling properties, single site
- Docker Swarm
  - Container-based (instead of VMs)
  - Relatively simple setup, easy generation
- Google Kubernetes

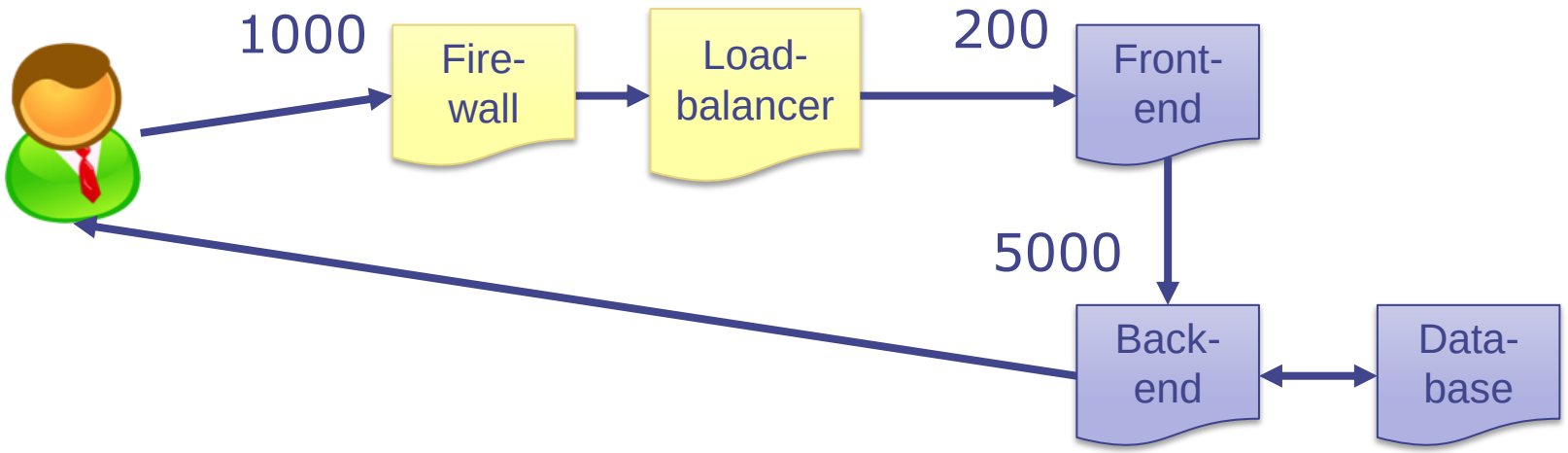
Docker/Kubernetes example: Guestbook application

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: redis-master
  # these labels can be applied automatically
  # from the labels in the pod template if not set
  # labels:
  #   app: redis
  #   role: master
  #   tier: backend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 1
  # selector can be applied automatically
  # from the labels in the pod template if not set
  # selector:
  #   matchLabels:
  #     app: guestbook
  #     role: master
  #     tier: backend
  template:
    metadata:
      labels:
        app: redis
        role: master
        tier: backend
    spec:
      containers:
      - name: master
        image: gcr.io/google_samples/gb-redisslave:v1
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 6379
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: redis-slave
  # these labels can be applied automatically
  # from the labels in the pod template if not set
  # labels:
  #   app: redis
  #   role: slave
  #   tier: backend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 2
  # selector can be applied automatically
  # from the labels in the pod template if not set
  # selector:
  #   matchLabels:
  #     app: guestbook
  #     role: slave
  #     tier: backend
  template:
    metadata:
      labels:
        app: redis
        role: slave
        tier: backend
    spec:
      containers:
      - name: slave
        image: gcr.io/google_samples/gb-redisslave:v1
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
        - name: GET_HOSTS_FROM
          value: dns
          # If your cluster config does not include a dns service, then to
          # instead access an environment variable to find the master
          # service's host, comment out the 'value: dns' line above, and
          # uncomment the line below.
          # value: env
        ports:
        - containerPort: 6379
---
apiVersion: v1
kind: Service
metadata:
  name: frontend
  # these labels can be applied automatically
  # from the labels in the pod template if not set
  # labels:
  #   app: guestbook
  #   tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  # selector can be applied automatically
  # from the labels in the pod template if not set
  # selector:
  #   matchLabels:
  #     app: guestbook
  #     tier: frontend
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
        - name: GET_HOSTS_FROM
          value: dns
          # If your cluster config does not include a dns service, then to
          # instead access environment variables to find service host
          # info, comment out the 'value: dns' line above, and uncomment the
          # line below.
          # value: env
        ports:
        - containerPort: 80
```

https://github.com/kubernetes/kubernetes/blob/release-1.2/examples/guestbook/all-in-one/guestbook-all-in-one.yaml

Templates, as they should be

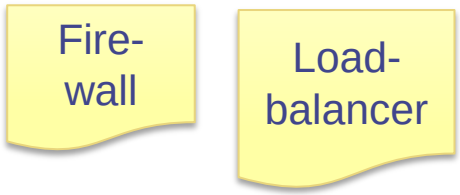
- **Quantitative** annotations
  - From profiling
- Scaling options
- …



Services?

- "Ordinary" application boxes?

- "Network" functions?

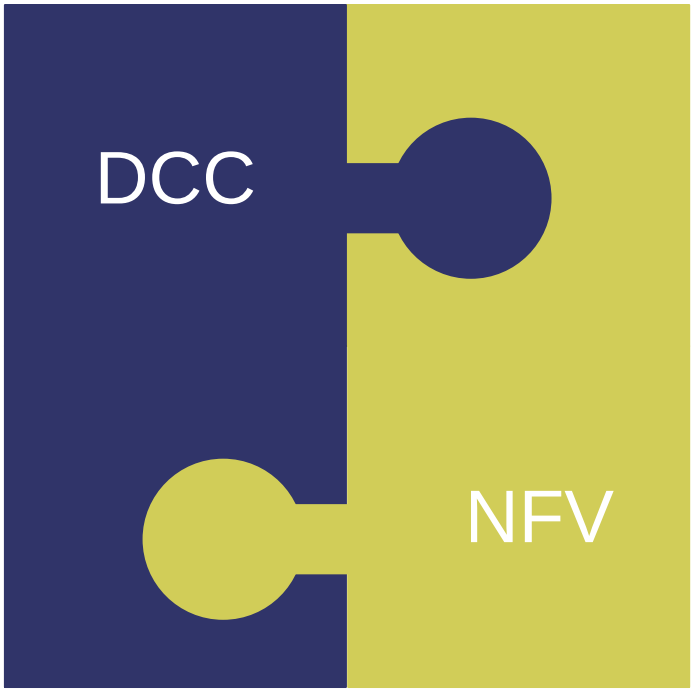


- What's the key, pivotal difference?

Claim: None!

- There is no inherent difference between
  - microservices
  - network functions
- Differences are a historical, ecosystem-based coincidence
  - E.g., separate orchestrators, …

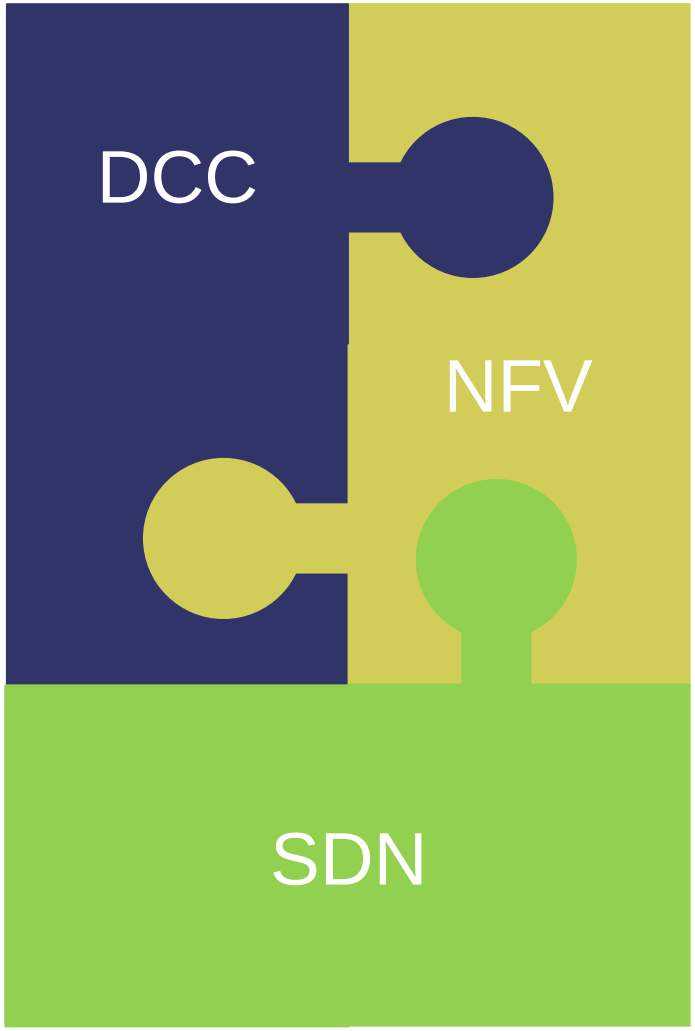
Mid-term goal: Merge NFV and (D)CC



Crossing from VNFs to SDN?

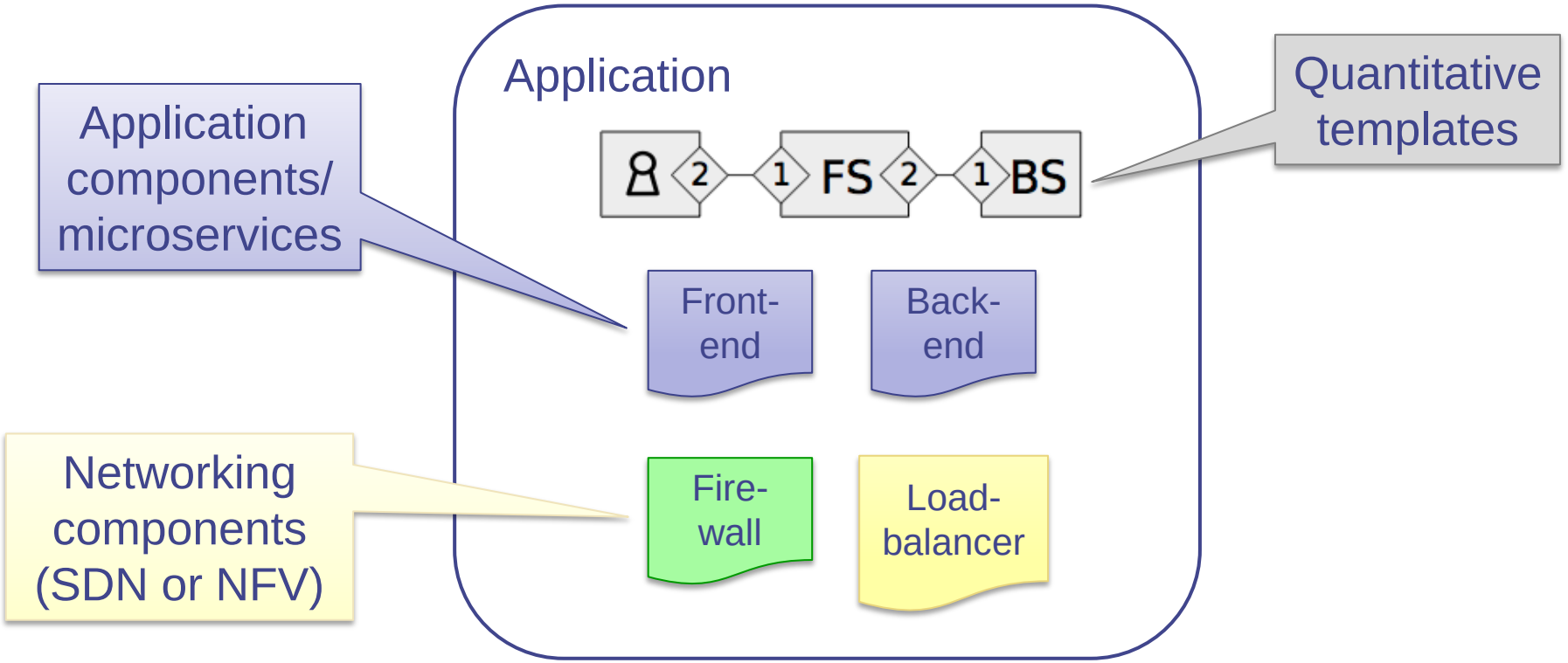
- Example: Load balancer
  - Given as a VNF
  - Or as SDN rules
- Decide which one?
- Automatically convert?
- Open!

Mid-term goal: Merge NFV, DCC, SDN



Applications become more complex

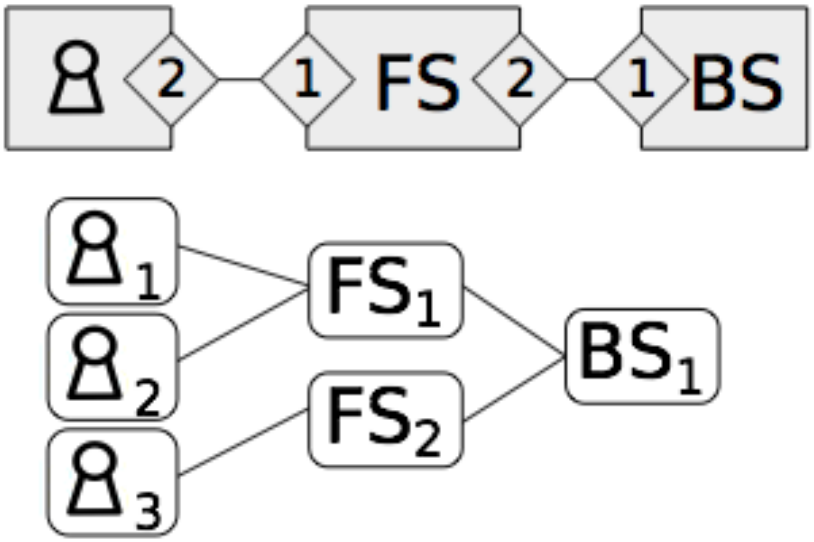
- **Application =**
  Actual application + networking

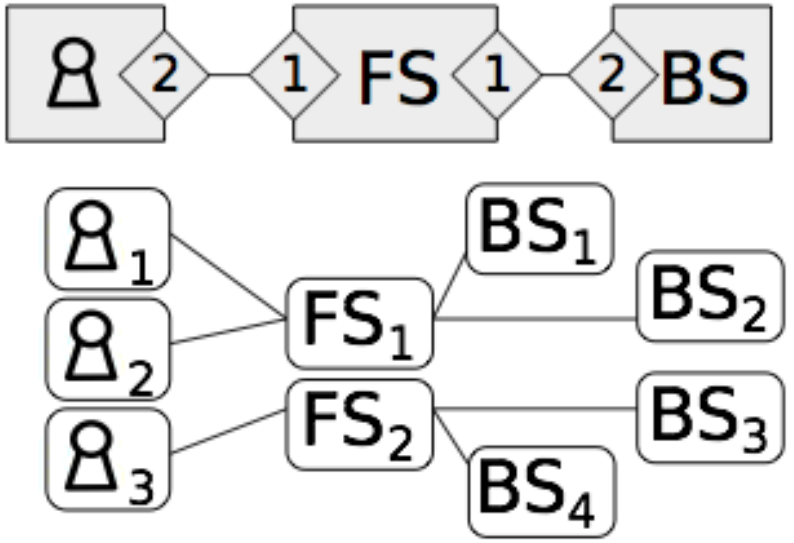


Remember

Developer knows best!

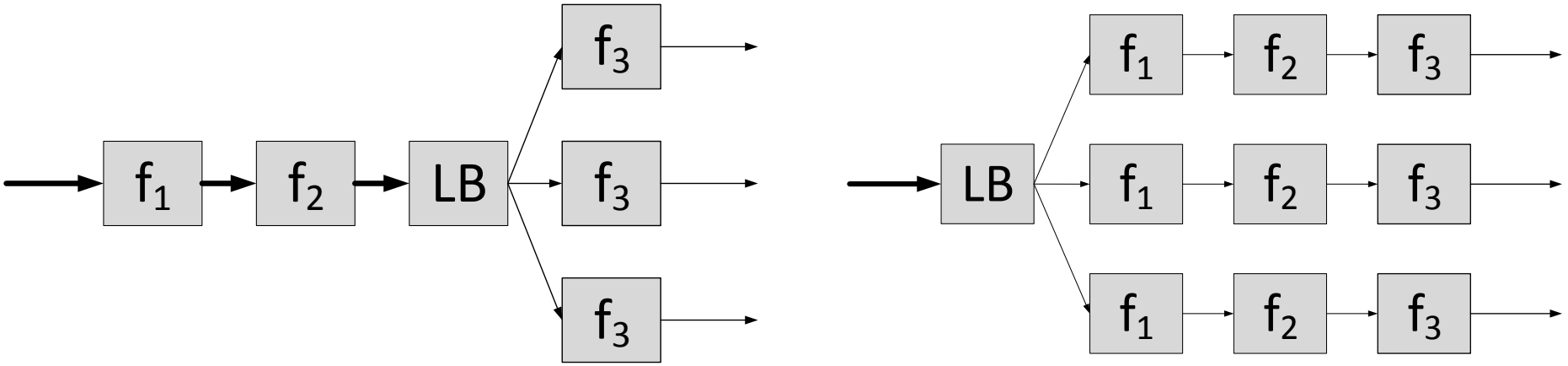
Example template: Quantitative scaling





M. Keller, C. Robberts, H. Karl: Template Embedding: Using Application Architecture to Allocate Resources in Distributed Clouds, UCC 2014

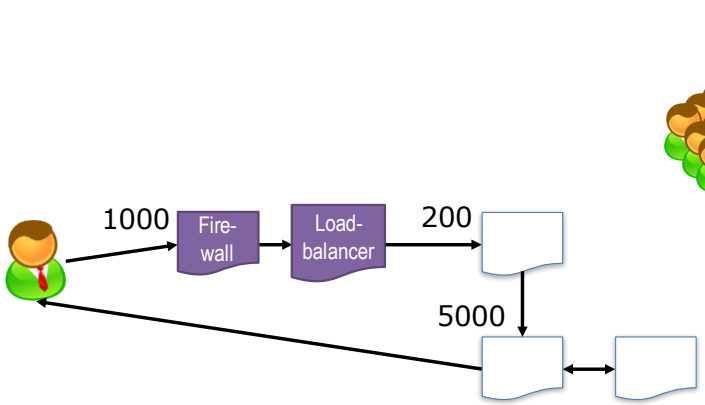
Example template: Reordering

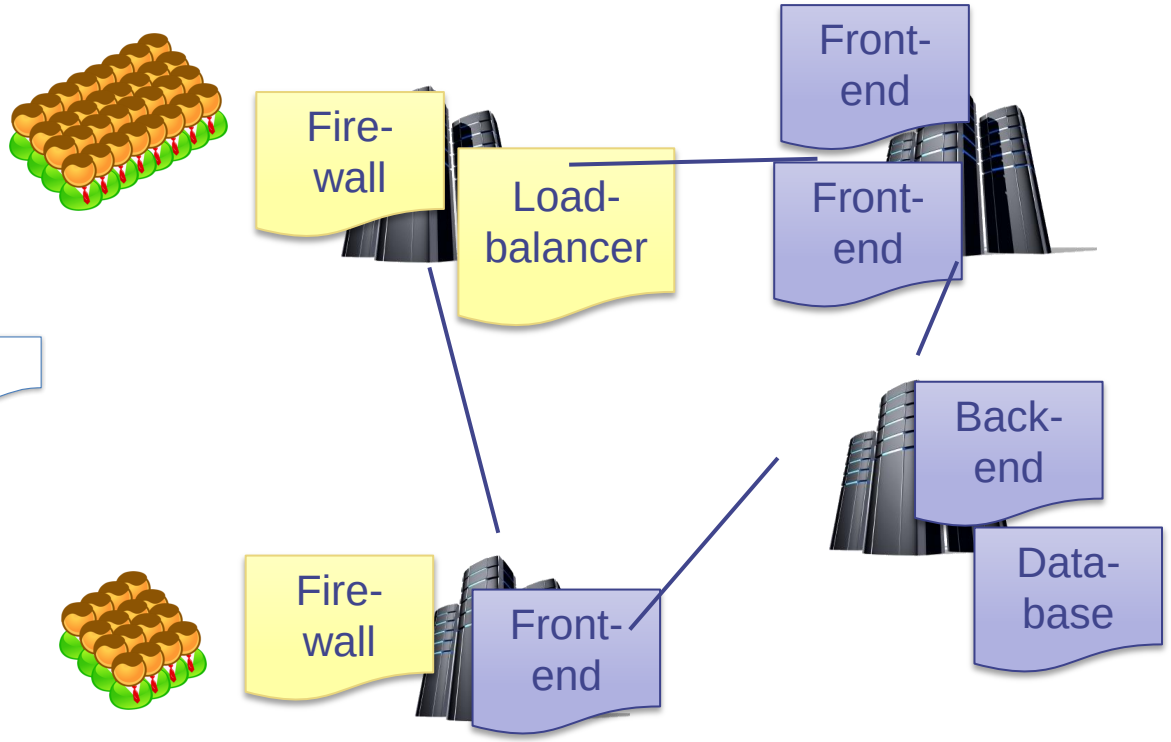


S. Mehraghdam, M. Keller, H. Karl: Specifying and Placing Chains of Virtual Network Functions, CloudNet 2014

Issues


Describe
Deploy
Configure


1000
Fire-
wall
Load-
balancer
200
5000


Fire-
wall
Load-
balancer
Front-
end
Front-
end
Back-
end
Data-
base
Fire-
wall
Front-
end

Deploying?

- Scaling?
- Placement?
- Lifecycle management?

- When to start, stop, migrate state of a component

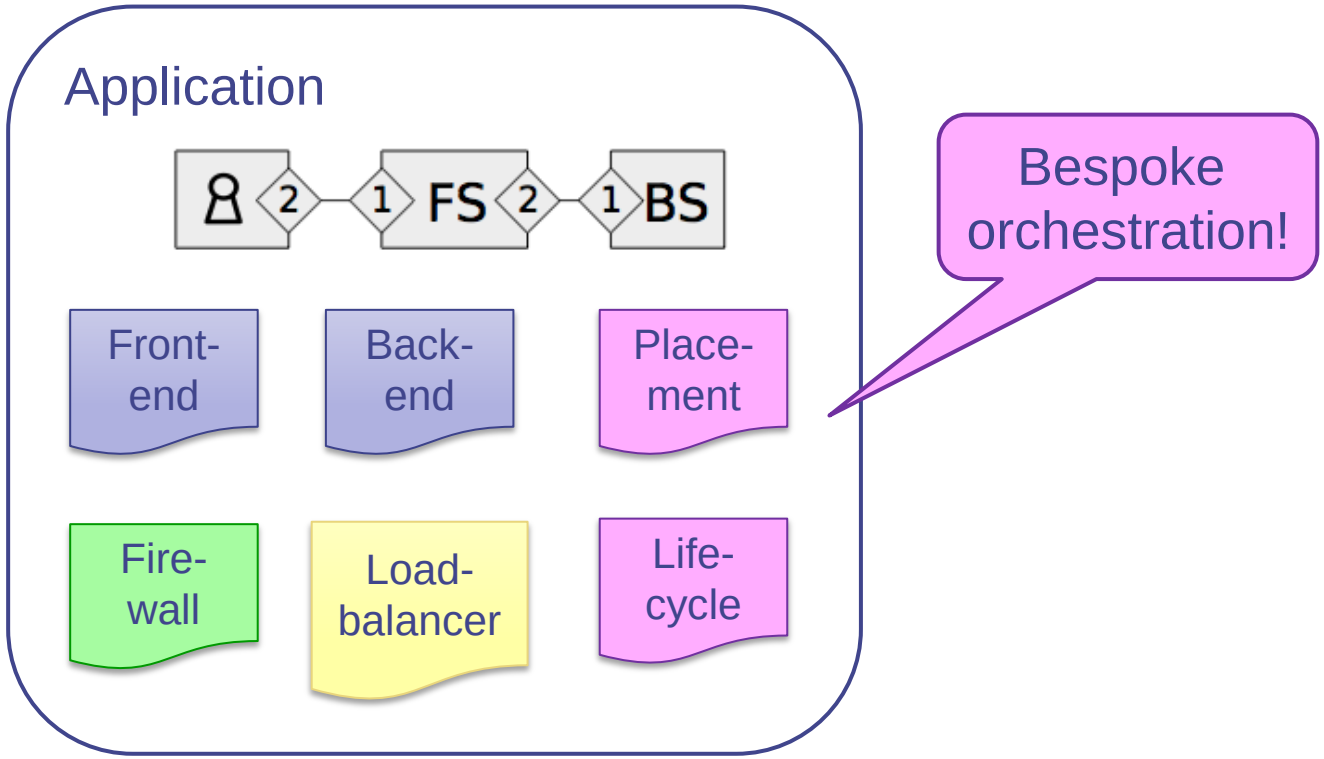
- Where to put which component?
- Virtual network embedding – NP-hard
- Embedding **scalable** templates: generalization
  - Still solvable with good heuristics

One-size-fits-all solution?

- Use a single, “perfect” deployment scheme?
  - Independent of application?
- Remember: Developer knows best!
  - SotA: Some parameterization exists, but limited in flexibility
    - E.g., Kubernetes
- No! Need bespoke deployment, orchestration schemes!

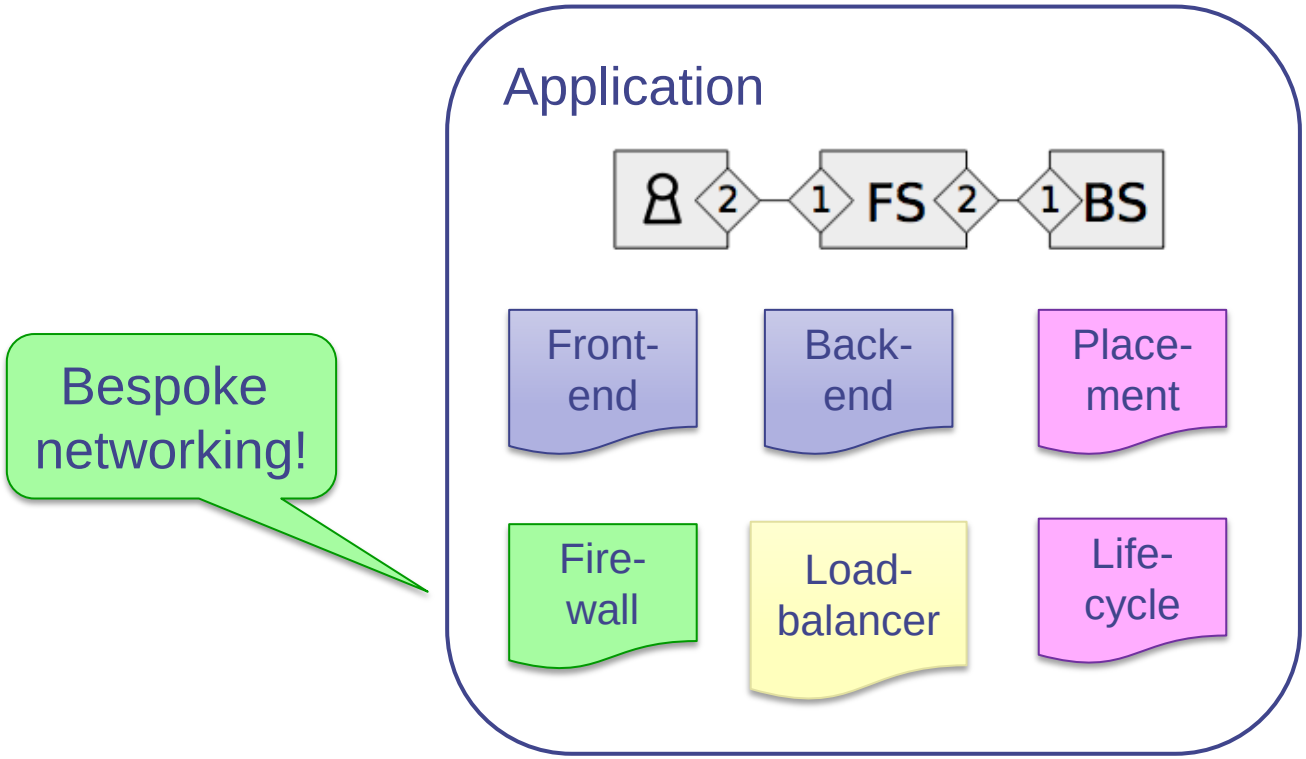
Developer knows best!

- **Application =**
  Orchestration + actual application + networking
  - UNIFY, SONATA, ...

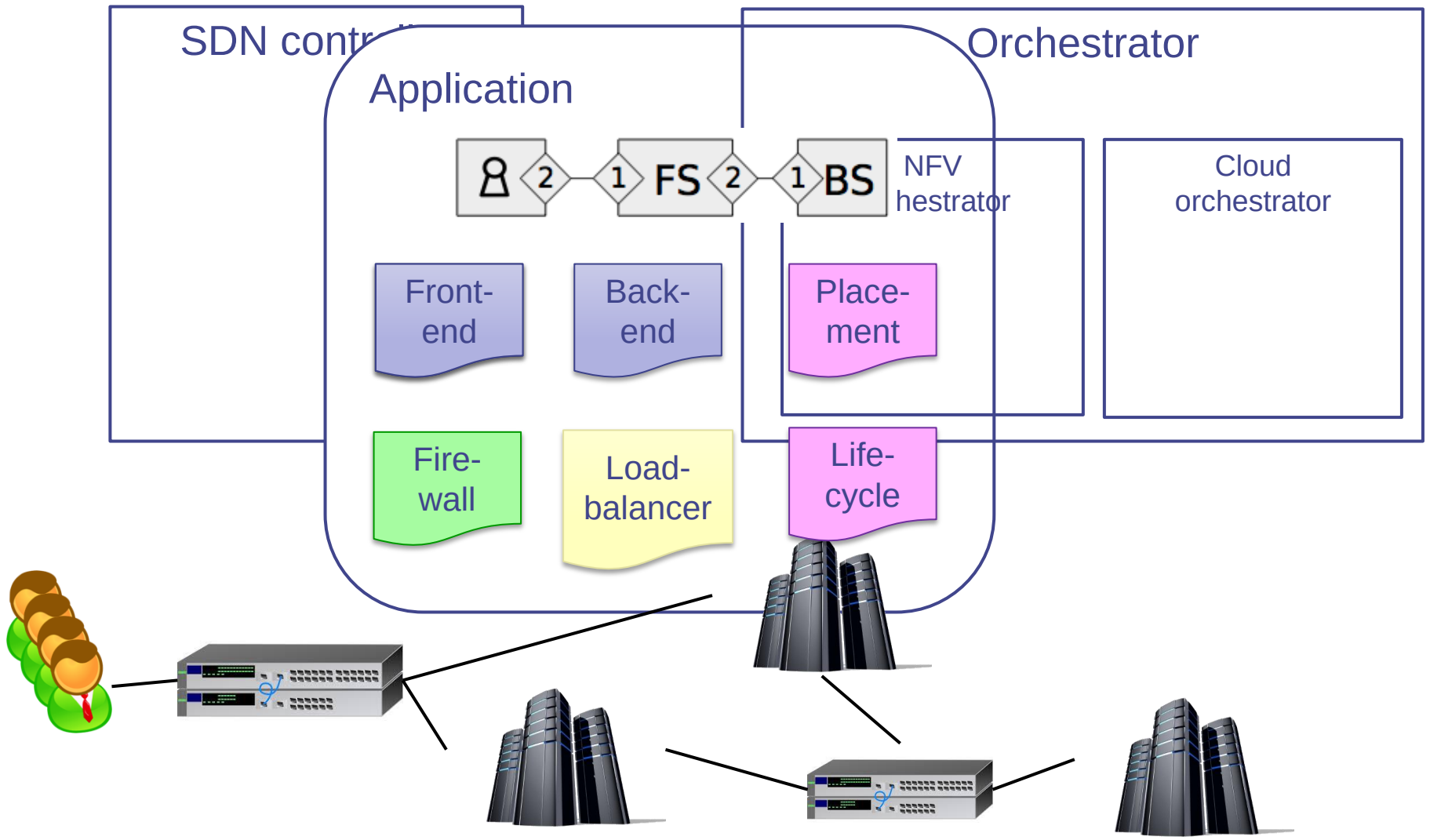


Reality check: Networking

- Networking: SDN controller applications
- Create network topology for application
- Developer support? Legacy?



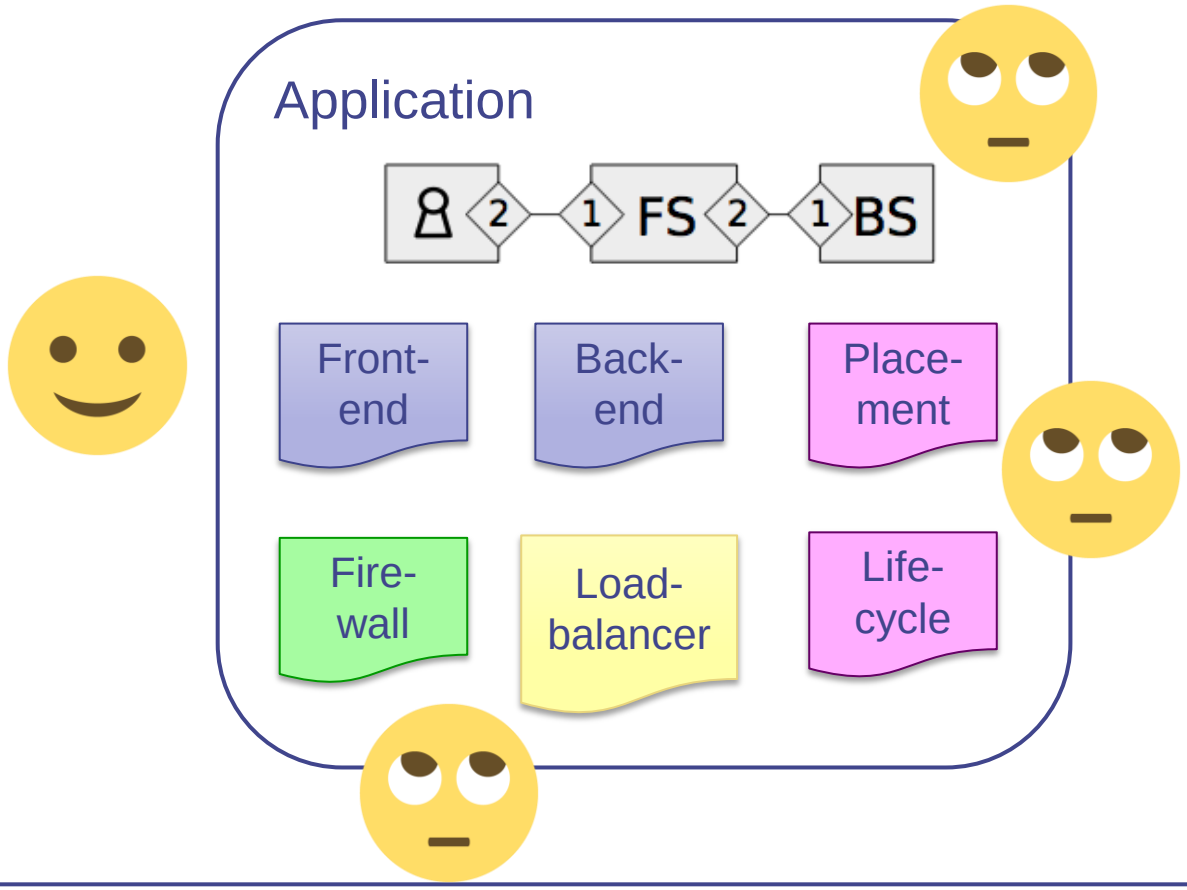
High-level architecture



Reality check: Developer

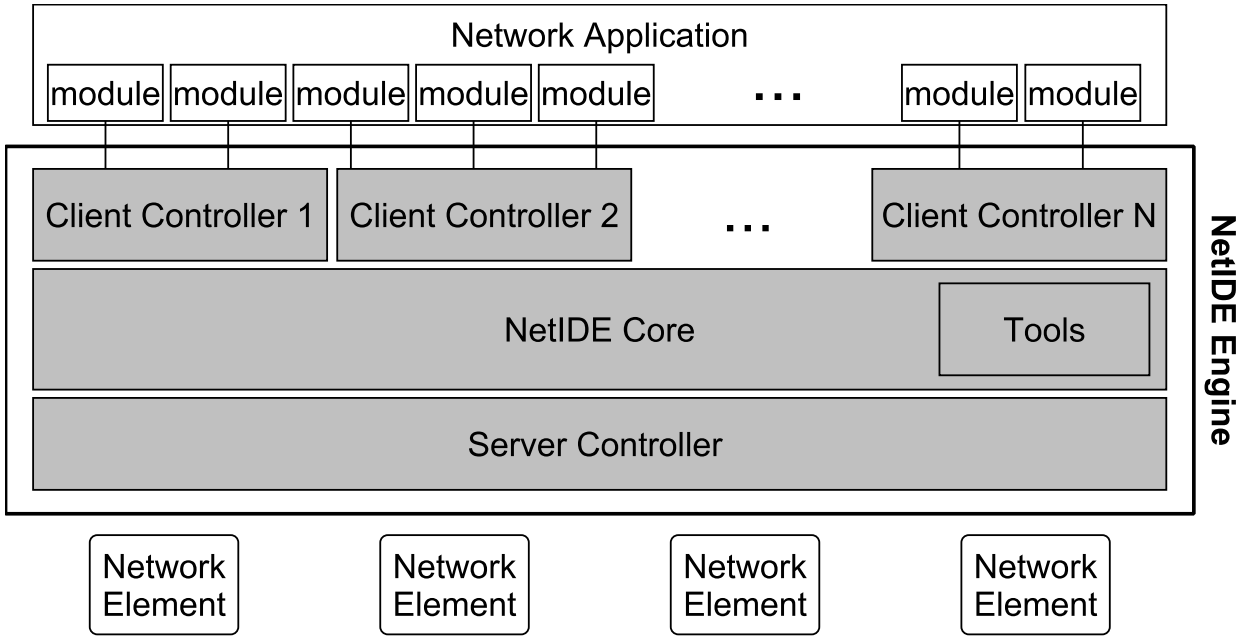
Developer knows best?

Developer knows best?



Reality check: Develop SDN network applications

- NetIDE: Bring SDN closer to developers
- Supporting legacy applications on controller frameworks

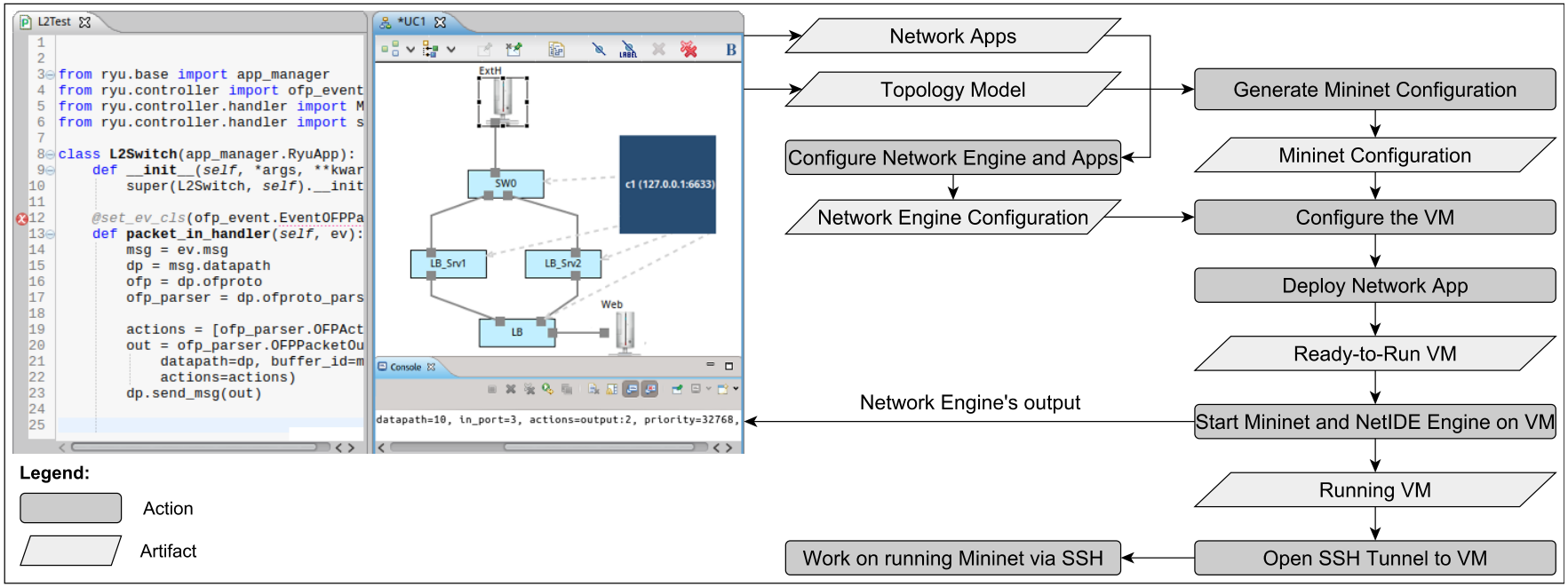


NetIDE

http://www.netide.eu

P. A. Aranda Gutiérrez et al., NetIDE: All-in-one framework for next generation, composed SDN applications, NetSoft Demo, 2016

NetIDE: Ease developer's tasks by SDK



P. A. Aranda Gutiérrez et al., NetIDE: All-in-one framework for next generation, composed SDN applications, NetSoft Demo, 2016

Reality check: Templates

- Ideal developer writes perfect templates
- Real developer?

- Option: semi-automatic generation?
  - Based on semantic understanding of components

SFB 901

ON - THE - FLY COMPUTING

https://sfb901.uni-paderborn.de

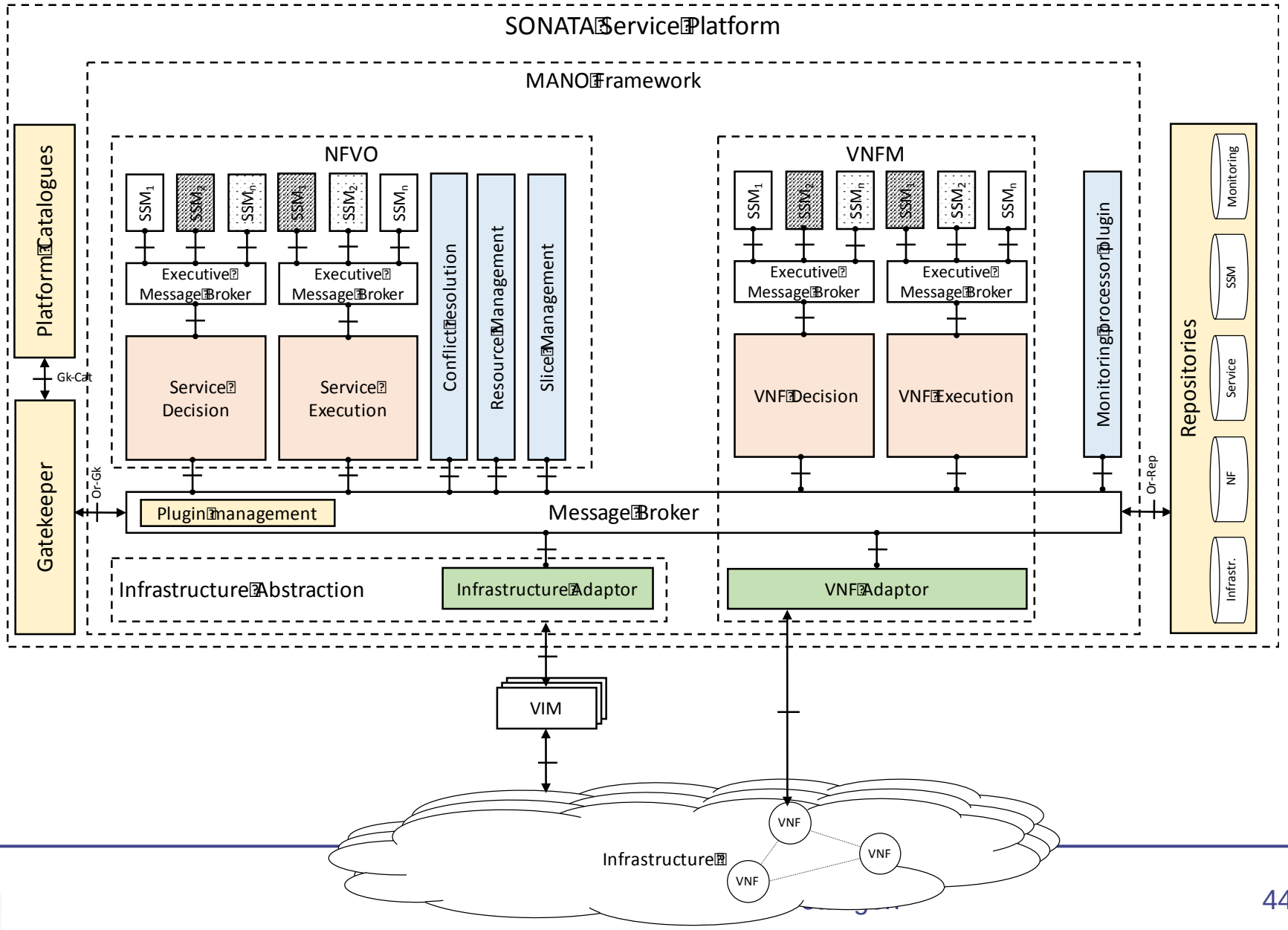
Reality check: Orchestration with 3rd party modules?

- Run code from outside application inside an orchestration platform?
  - Security, isolation, conflicts, …?

- Sure – as microservices!
  - SONATA (5G PPP): microservices-based orchestration

http://sonata-nfv.eu

SONATA: Architecture overview



Challenge: Education!

- Teaching curricula: Integrate networking, distributed systems, virtualization
- Continuous education

Conclusion

- NFV and (D)CC need to merge; SDN integrated
- Developers and Ops need to understand that
  - DevOps for complex, distributed applications
- Research & education task

NetIDE

sonata

SFB 901
ON - THE - FLY COMPUTING