# Automatic Application Deployment in the Cloud: from Practice to Theory and Back

**Gianluigi Zavattaro** — University of Bologna - Italy; FoCUS research team INRIA - France

Based on joint work with:

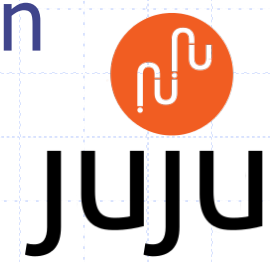
**T.A. Lascu, J. Mauro** — Univ. of Bologna - Italy

**R. Di Cosmo, S. Zacchiroli, J. Zwolakowski** — Univ. Paris Diderot - France

**A. Eiche** — Mandriva SA - France

# Cloud applications

- Cloud computing offers the possibility to build **sophisticated** software systems on virtualized infrastructures at a fraction of the **time / cost** necessary just few years ago
- We can give a look at Juju to have an idea of how cloud applications can be **easily** deployed nowadays



# Cloud application management is any way a complex task

- Even if tool-supported, the cloud application **operator** must decide:
  - which software **components** to select
  - the overall application **architecture**
  - the order of the **configuration** actions
  - ...

# The challenge

- Understand how much of the operator's activities can be **automatised**:
  - selection of the software components (selected from appropriate **repositories**)
  - **synthesis** of the overall architecture
  - **planning** of the configuration actions to be executed to realize the expected architecture
  - …

# Our current “practical” results

- Definition of a language for describing component’s **repositories**

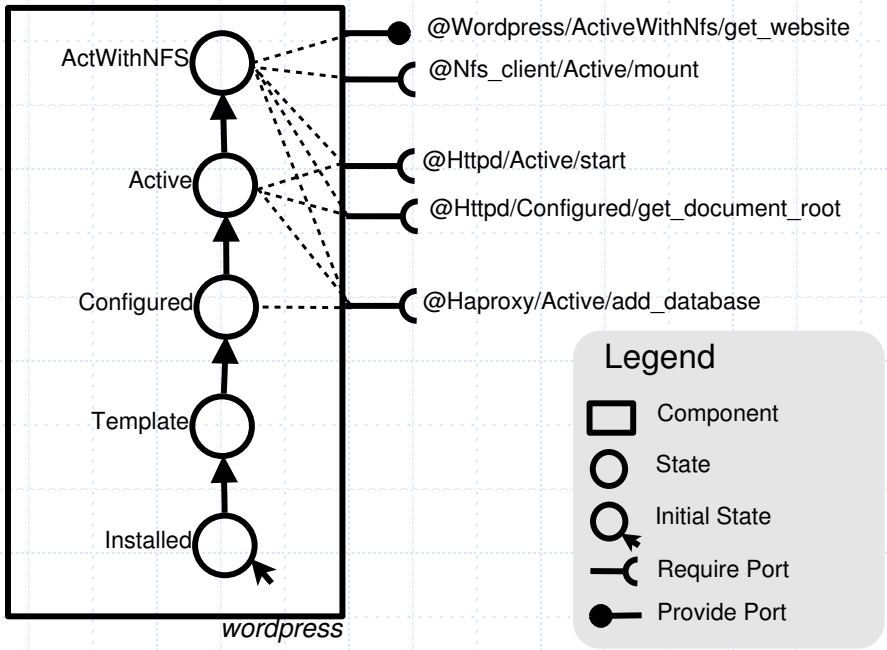
```
{
  "states": [
   {
    "provide": {},
    "require": {},
    "initial": true,
    "name": "Installed",
    "successors": [
     "Template"
    ]
   },
   {
    "provide": {},
    "require": {},
    "successors": [
     "Configured"
    ],
    "name": "Template"
   },
```

```
{
 "provide": {},
 "require": {
  "@Haproxy/Active/add_database": 1
 },
 "successors": [
  "Active"
 ],
 "name": "Configured"
},
{
 "provide": {},
 "require": {
  "@Haproxy/Active/add_database": 1,
  "@Httpd/Active/start": 1,
  "@Httpd/Configured/get_document_root": 1
 },
 "successors": [
  "ActiveWithNfs"
 ],
 "name": "Active"
},
```

```
{
 "provide": {
  "@Wordpress/ActiveWithNfs/get_website": 1000
 },
 "require": {
  "@Haproxy/Active/add_database": 1,
  "@Httpd/Active/start": 1,
  "@Httpd/Configured/get_document_root": 1,
  "@Nfs_client/Active/mount": 1
 },
 "name": "ActiveWithNfs"
}
],
"name": "Wordpress"
}
```

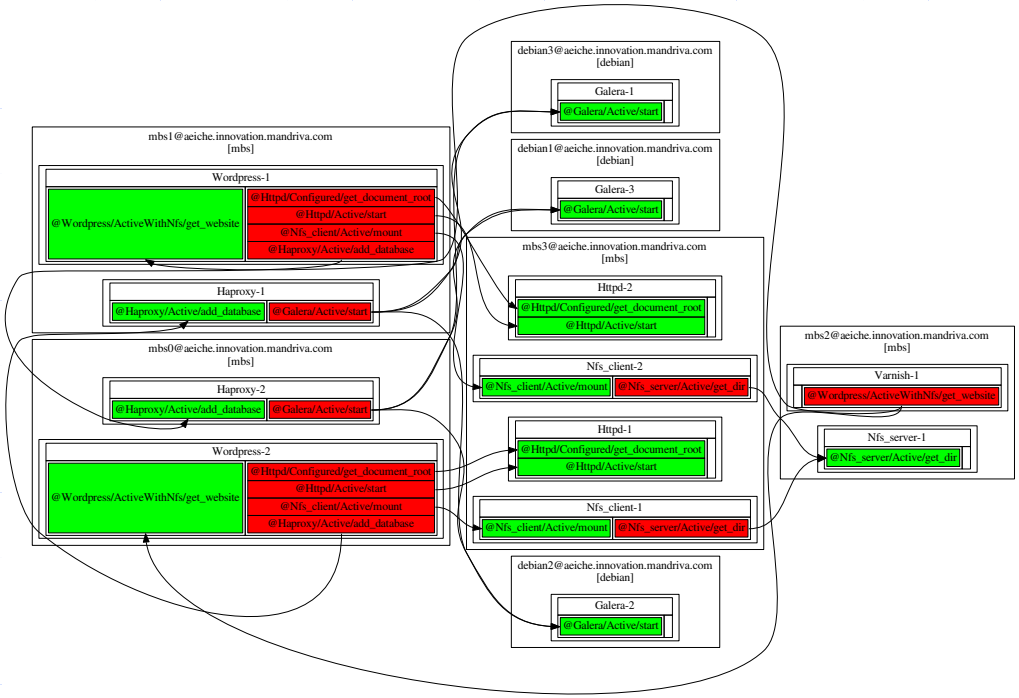
# Our current “practical” results

- Definition of a language for describing component’s **repositories**



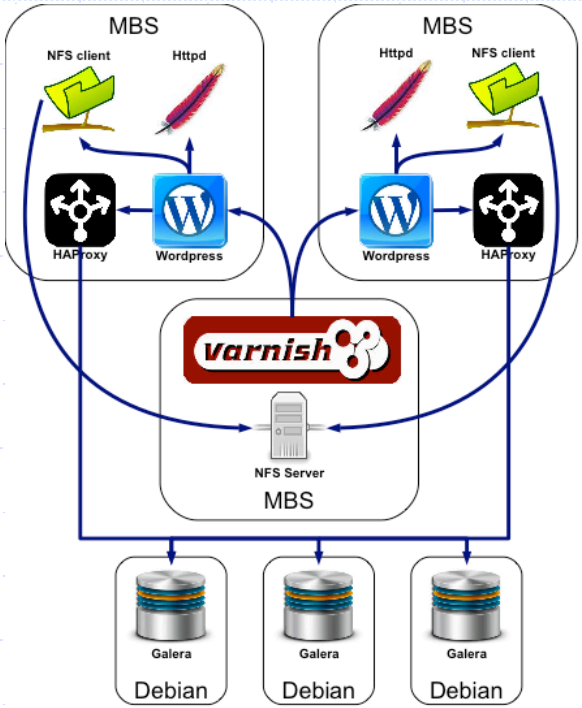
# Our current “practical” results

- Realization of a tool for component’s selection and architecture **synthesis**



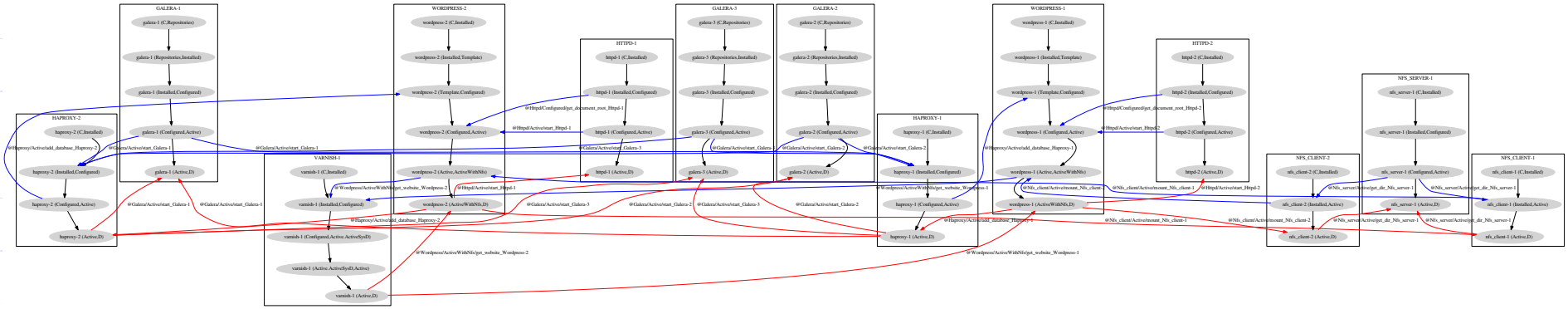
# Our current "practical" results

- Realization of a tool for component's selection and architecture **synthesis**



# Our current “practical” results

- Realization of a tool for **planning** the configuration actions to be executed

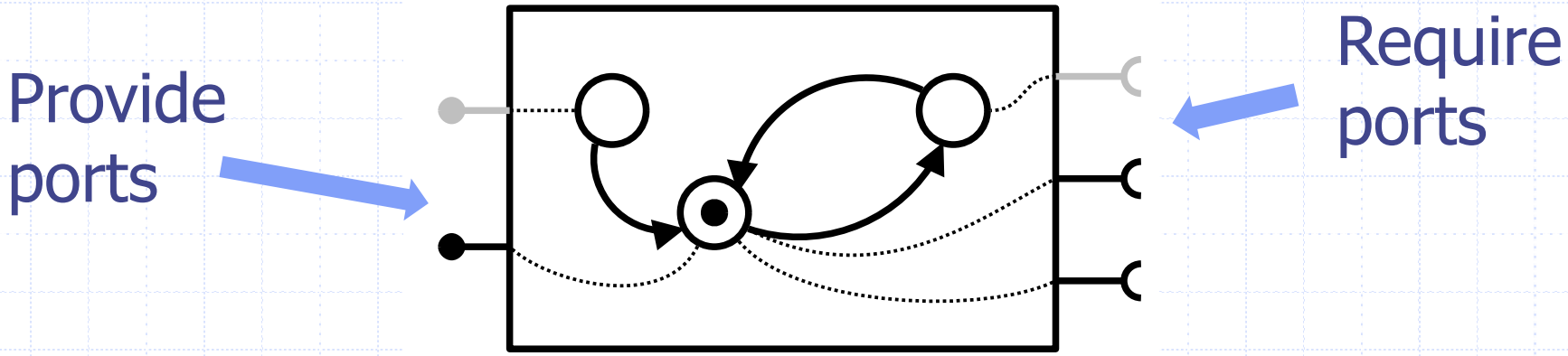


# Structure of the talk

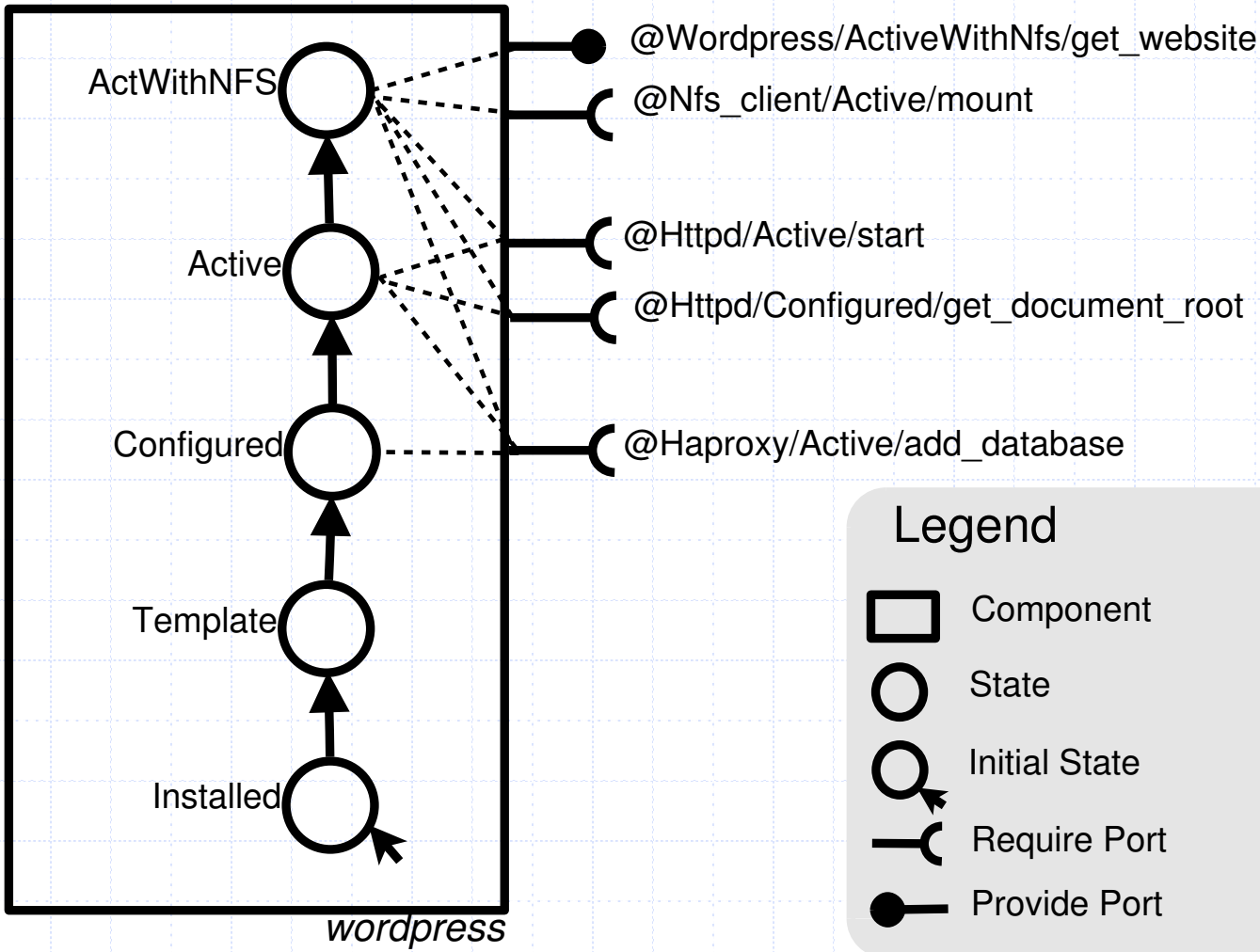
- **Formalizing the deployment problem**
- Decidability/complexity results
- Fully automatic deployment
  (without capacity constraints and conflicts)
- Constraints and conflicts strike back
- Conclusion and Open issues
- Related work

# Component types

- A component has **provide** and **require** ports
- A component has an internal **state machine**
- Ports are **active** or **inactive** according to the current internal state

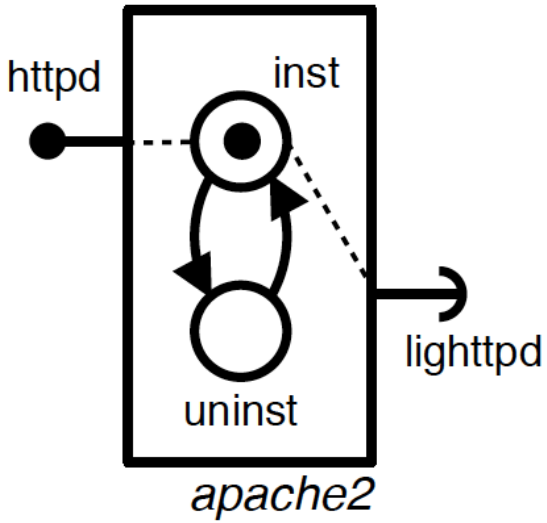


# Example: the Wordpress component type



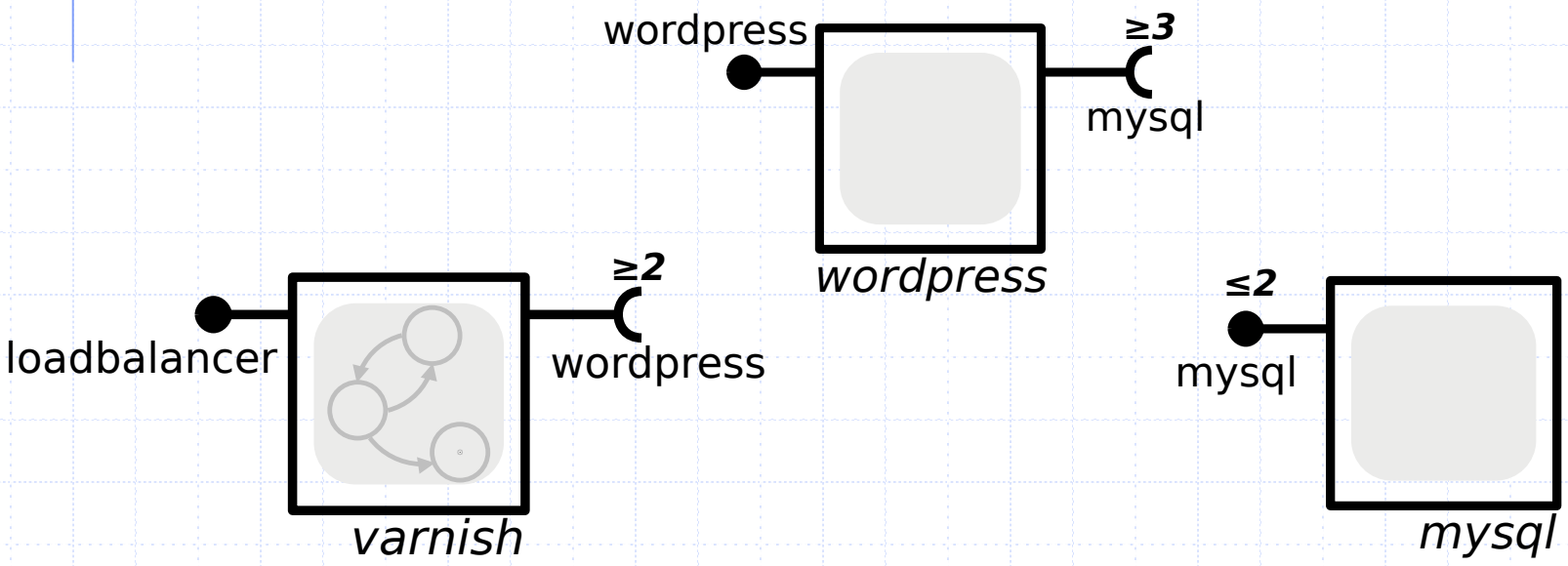
# Conflicts

- Conflicts are expressed as **special** ports
  - The apache web server is in **conflict** with the lighttpd web server



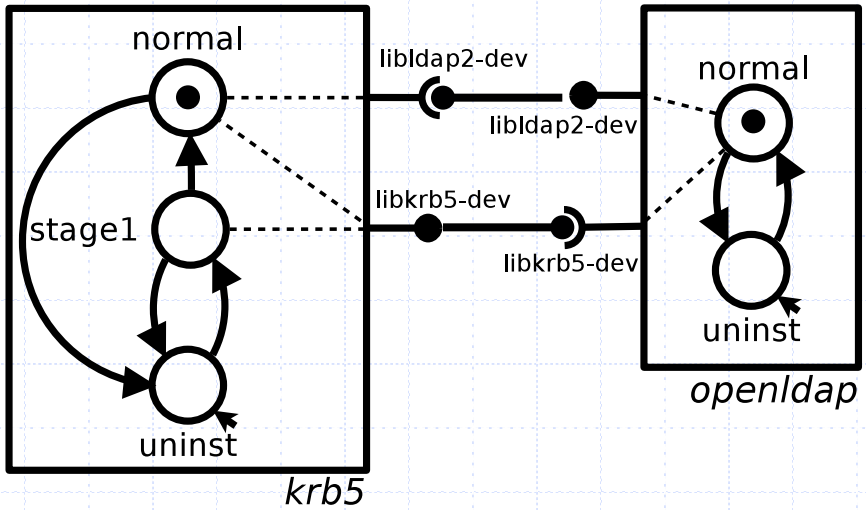
# Capacity constraints

- Provide (resp. require) ports could have an associated **upper** (resp. **lower**) **bound** to the number of connections



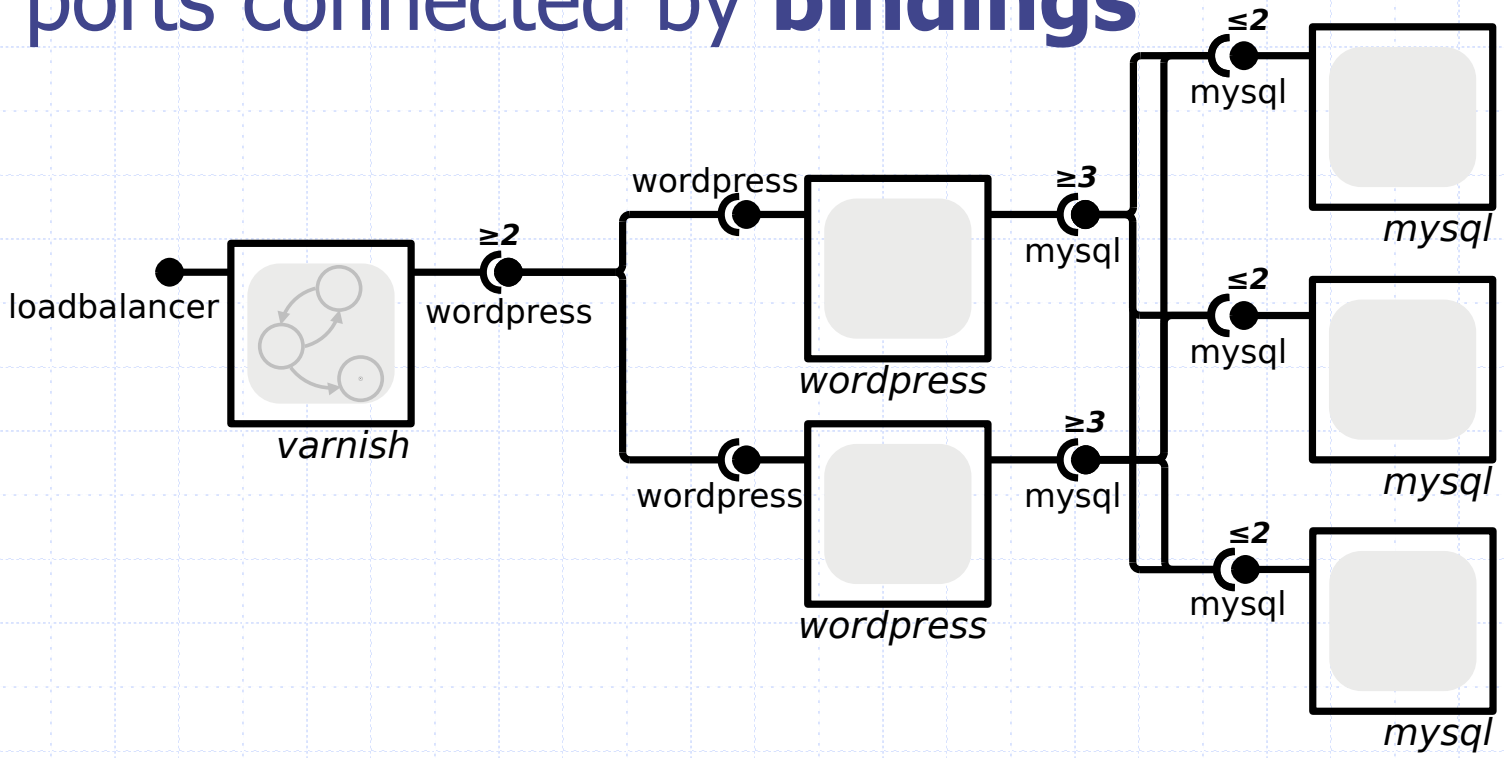
# Configurations

- Component **instances**, with a current **state**, and complementary provide/require ports connected by **bindings**
  - Example: Kerberos with ldap support in Debian (example of **circular** dependency)



# Configurations

- Component **instances**, with a current **state**, and complementary provide/require ports connected by **bindings**



# Formalizing the "deployment" problem

▶ **Definition 1** (Component type). The set $\Gamma$ of *component types* of the Aeolus model, ranged over by $\mathcal{T}_1, \mathcal{T}_2, \ldots$ contains 5-ple $\langle Q, q_0, T, P, D \rangle$ where:

- $Q$ is a finite set of states;
- $q_0 \in Q$ is the initial state and $T \subseteq Q \times Q$ is the set of *transitions*;
- $P = \langle \mathbf{P}, \mathbf{R} \rangle$, with $\mathbf{P}, \mathbf{R} \subseteq \mathcal{I}$, is a pair composed of the set of *provide* and the set of *require*-ports, respectively;
- $D$ is a function from $Q$ to 2-ple in $(\mathbf{P} \nrightarrow \mathbb{N}^+_\infty) \times (\mathbf{R} \nrightarrow \mathbb{N})$.

▶ **Definition 2** (Configuration). A *configuration* $\mathcal{C}$ is a quadruple $\langle U, Z, S, B \rangle$ where:

- $U \subseteq \Gamma$ is the finite *universe* of all available component types;
- $Z \subseteq \mathcal{Z}$ is the set of the currently deployed *components*;
- $S$ is the component *state description*, i.e., a function that associates to components in $Z$ a pair $\langle \mathcal{T}, q \rangle$ where $\mathcal{T} \in U$ is a component type $\langle Q, q_0, T, P, D \rangle$, and $q \in Q$ is the current component state;
- $B \subseteq \mathcal{I} \times Z \times Z$ is the set of *bindings*, namely 3-ples composed by an interface, the component that requires that interface, and the component that provides it; we assume that the two components are distinct.

# Formalizing the "deployment" problem

- **Definition 5** (Actions). The set $\mathcal{A}$ contains the following actions:
  - $stateChange(z, q_1, q_2)$ where $z \in \mathcal{Z}$: change the state of the component $z$ from $q_1$ to $q_2$;
  - $bind(r, z_1, z_2)$ where $z_1, z_2 \in \mathcal{Z}$ and $r \in \mathcal{I}$: add a binding between $z_1$ and $z_2$ on port $r$;
  - $unbind(r, z_1, z_2)$ where $z_1, z_2 \in \mathcal{Z}$ and $r \in \mathcal{I}$: remove the specified binding;
  - $new(z : \mathcal{T})$ where $z \in \mathcal{Z}$ and $\mathcal{T}$ is a component type: add a new component $z$ of type $\mathcal{T}$;
  - $del(z)$ where $z \in \mathcal{Z}$: remove the component $z$ from the configuration.

# Formalizing the "deployment" problem

▶ **Definition 6** (Reconfigurations). Reconfigurations are denoted by transitions $\mathcal{C} \xrightarrow{\alpha} \mathcal{C}'$ meaning that the execution of $\alpha \in \mathcal{A}$ on the configuration $\mathcal{C}$ produces a new configuration $\mathcal{C}'$. The transitions from a configuration $\mathcal{C} = \langle U, Z, S, B \rangle$ are defined as follows:

$$\mathcal{C} \xrightarrow{stateChange(z,q_1,q_2)} \langle U, Z, S', B \rangle$$
$$\text{if } \mathcal{C}[z].\texttt{state} = q_1$$
$$\text{and } (q_1, q_2) \in \mathcal{C}[z].\texttt{trans}$$
$$\text{and } S'(z') = \begin{cases} \langle \mathcal{C}[z].\texttt{type}, q_2 \rangle & \text{if } z' = z \\ \mathcal{C}[z'] & \text{otherwise} \end{cases}$$

$$\mathcal{C} \xrightarrow{new(z:\mathcal{T})} \langle U, Z \cup \{z\}, S', B \rangle$$
$$\text{if } z \notin Z, \mathcal{T} \in U$$
$$\text{and } S'(z') = \begin{cases} \langle \mathcal{T}, \mathcal{T}.\texttt{init} \rangle & \text{if } z' = z \\ \mathcal{C}[z'] & \text{otherwise} \end{cases}$$

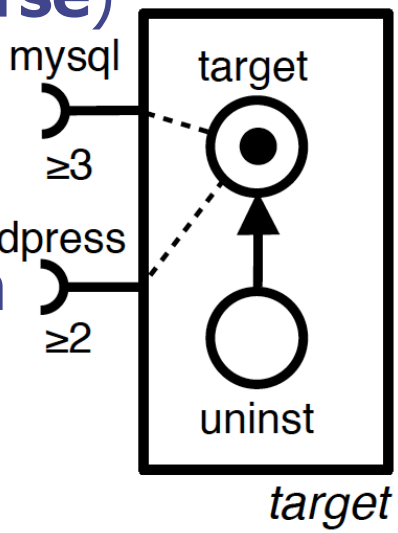
$$\mathcal{C} \xrightarrow{bind(r,z_1,z_2)} \langle U, Z, S, B \cup \langle r, z_1, z_2 \rangle \rangle$$
$$\text{if } \langle r, z_1, z_2 \rangle \notin B$$
$$\text{and } r \in \mathcal{C}[z_1].\texttt{req} \cap \mathcal{C}[z_2].\texttt{prov}$$

$$\mathcal{C} \xrightarrow{del(z)} \langle U, Z \setminus \{z\}, S', B' \rangle$$
$$\text{if } S'(z') = \begin{cases} \bot & \text{if } z' = z \\ \mathcal{C}[z'] & \text{otherwise} \end{cases}$$
$$\text{and } B' = \{\langle r, z_1, z_2 \rangle \in B \mid z \notin \{z_1, z_2\}\}$$

$$\mathcal{C} \xrightarrow{unbind(r,z_1,z_2)} \langle U, Z, S, B \setminus \langle r, z_1, z_2 \rangle \rangle \quad \text{if } \langle r, z_1, z_2 \rangle \in B$$

# “Deployment” problem

- Input:
  - A set of component types (called **Universe**)
  - One **target** component type-state pair
- Output:
  - **Yes**, if there exists a **deployment plan**
  - **No**, otherwise

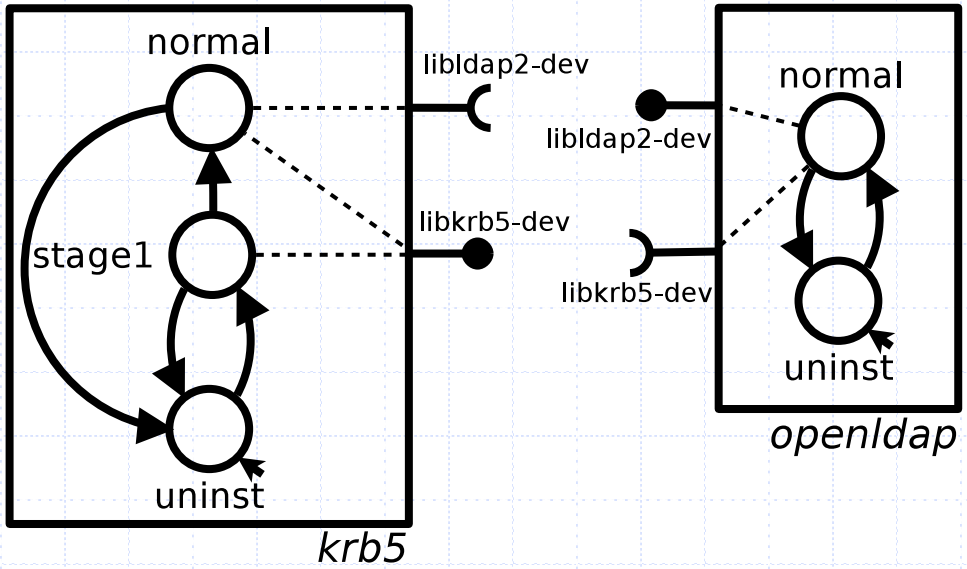



**Deployment plan**:

a sequence of actions leading to a final configuration containing at least one component of the given target type, in the given target state

# Deployment problem: example

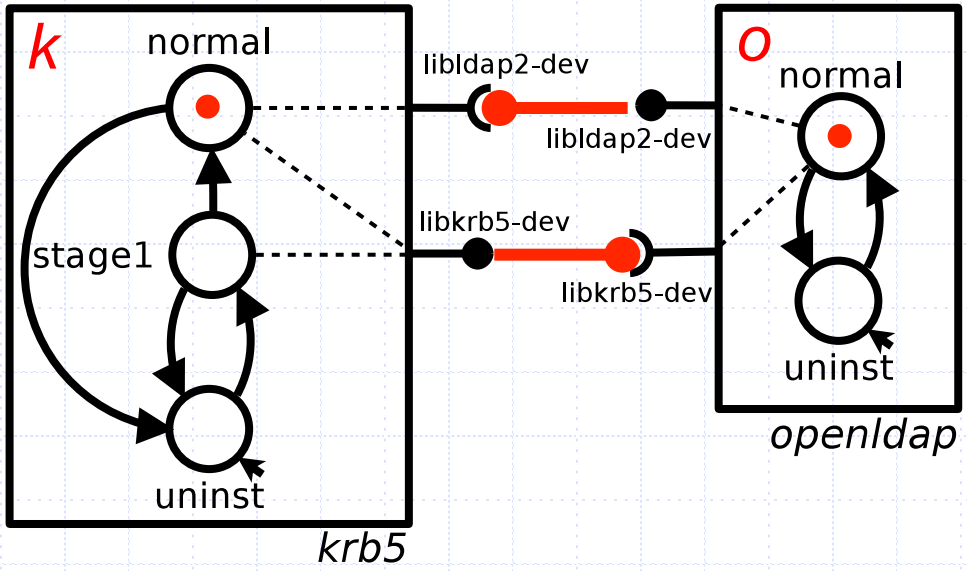
- Consider the problem of installing kerberos with ldap support in Debian
  - **Universe**: packages krb5 and openldap
  - **Target**: krb5 in normal state



# Deployment problem: example

- Deployment plan:

```
new(k:krb5),new(o:openldap),
stateChange(k,uninst,stage1),
bind(libkrb5-dev,o,k),stateChange(o,uninst,normal),
bind(libldap2-dev,k,o),
stateChange(k,stage1,normal)
```



# Structure of the talk

- Formalizing the deployment problem
- **Decidability/complexity results**
- Fully automatic deployment
  (without capacity constraints and conflicts)
- Constraints and conflicts strike back
- Conclusion and Open issues
- Related work

# Summary of decidability/complexity results

| Component model | Deployment is |
|---|---|
| Full component model | Undecidable [SEFM12] |
| No capacity constraints | Ackermann-hard [ICALP13,I&C14] |
| No capacity constraints, No conflicts | Quadratic [SEFM12] |

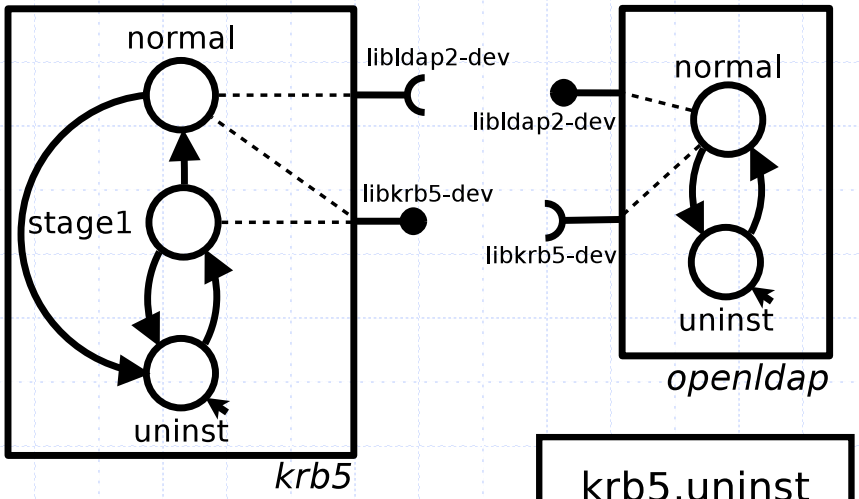
# Quadratic algorithm without constraints and conflicts [SEFM12]

- **Forward** reachability algorithm
  - all reachable states computed by saturation

**Algorithm 1** Checking achievability in the Aeolus$^-$ model

**function** ACHIEVABILITY($U$, $\mathcal{T}$, $q$)
  $absConf := \{\langle \mathcal{T}', \mathcal{T}'.\texttt{init} \rangle \mid \mathcal{T}' \in U\}$
  $provPort := \bigcup_{\langle \mathcal{T}', q' \rangle \in absConf} \{dom(\mathcal{T}'.\mathbf{P}(q'))\}$
  **repeat**
    $new := \{\langle \mathcal{T}', q' \rangle \mid \langle \mathcal{T}', q'' \rangle \in absConf, (q'', q') \in \mathcal{T}'.\texttt{trans}\} \setminus absConf$
    $newPort := \bigcup_{\langle \mathcal{T}', q' \rangle \in new} \{dom(\mathcal{T}'.\mathbf{P}(q'))\}$
    **while** $\exists \langle \mathcal{T}', q' \rangle \in new \;.\; dom(\mathcal{T}'.\mathbf{R}(q')) \not\subseteq provPort \cup newPort$ **do**
      $new := new \setminus \{\langle \mathcal{T}', q' \rangle\}$
      $newPort := \bigcup_{\langle \mathcal{T}', q' \rangle \in new} \{dom(\mathcal{T}'.\mathbf{P}(q'))\}$
    **end while**
    $absConf := absConf \cup new$
    $provPort := provPort \cup newPort$
  **until** $new = \emptyset$
  **if** $\langle \mathcal{T}, q \rangle \in absConf$ **then return true**
  **else return false**
  **end if**
**end function**

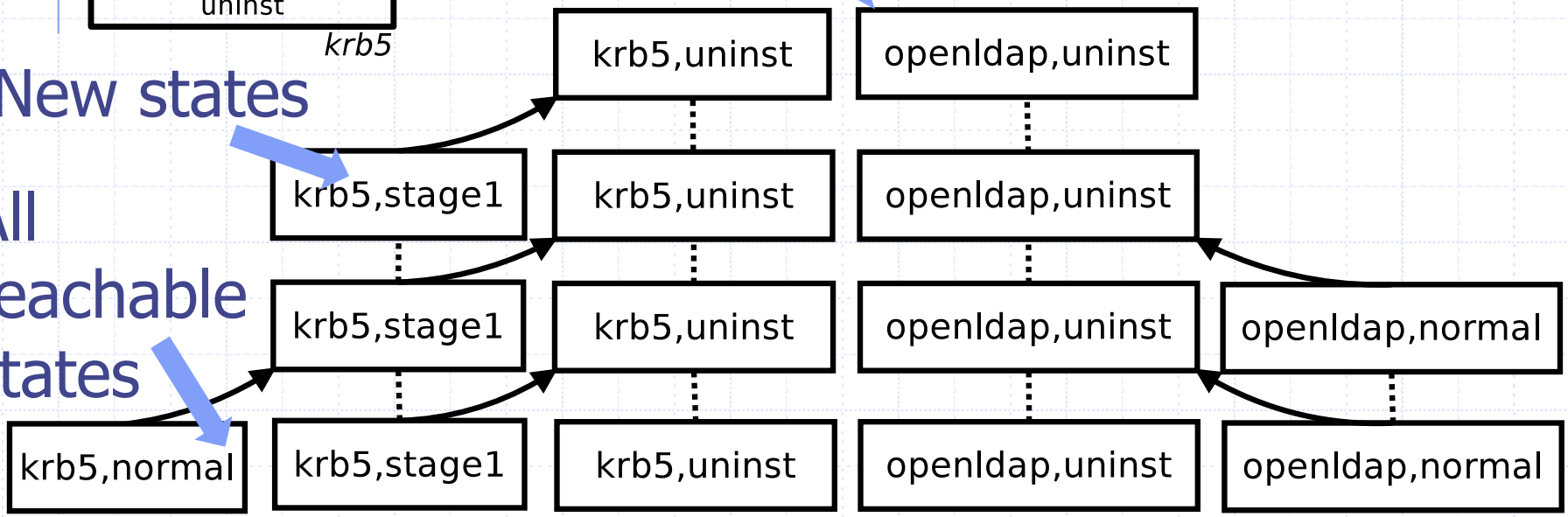
# Example: the kerberos case-study




Initial states

New states

All reachable states



# Lesson learned from the foundational study

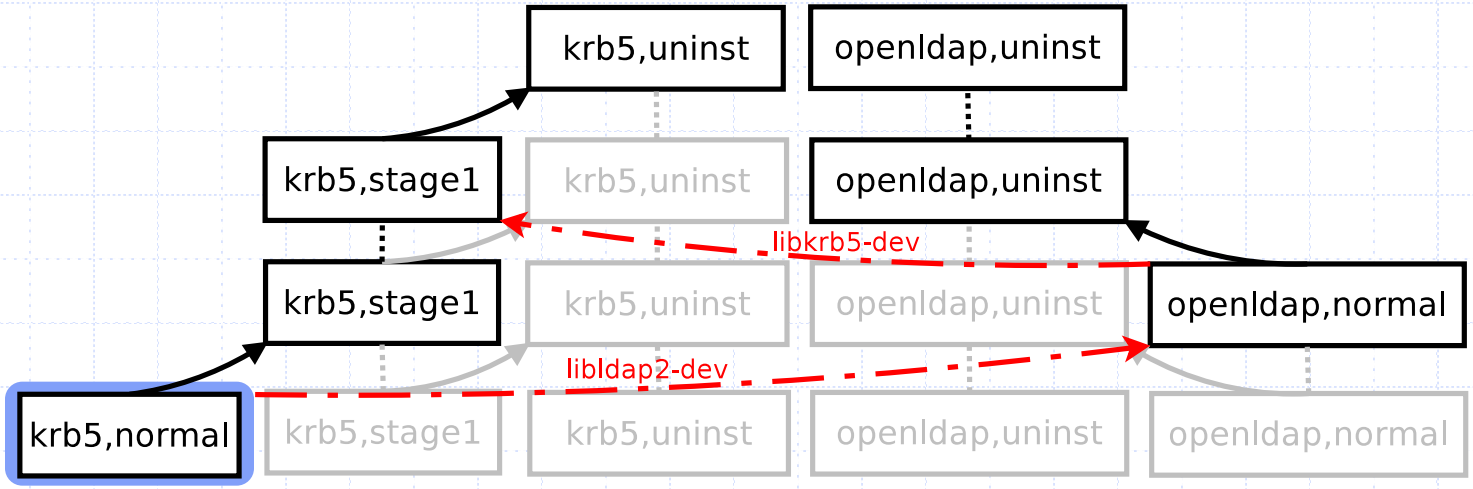
Deployment can be reasonably **fully automatised** if we do not consider capacity constraints and conflicts

# Structure of the talk

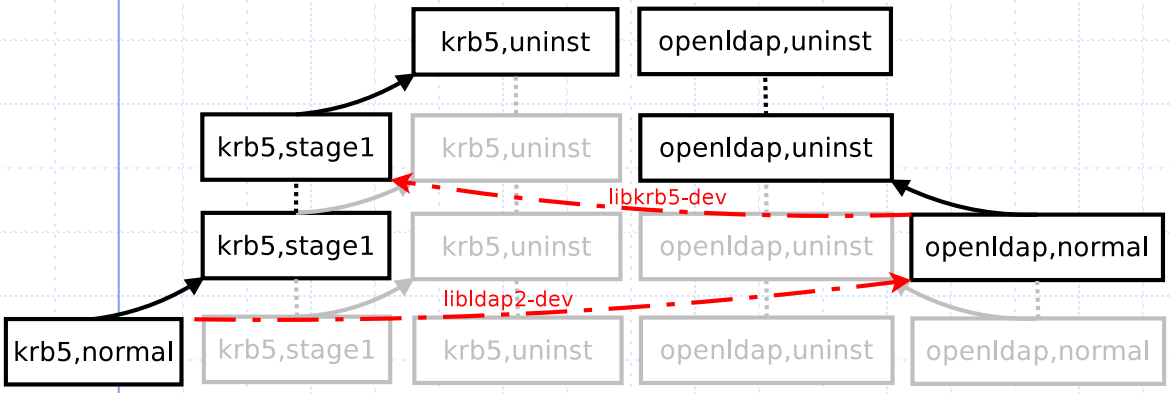
- Formalizing the deployment problem
- Decidability/complexity results
- **Fully automatic deployment**
  (without capacity constraints and conflicts)
- Constraints and conflicts strike back
- Conclusion and Open issues
- Related work

# Fully automated deployment (no capacity, no conflicts) [ICTAI13]

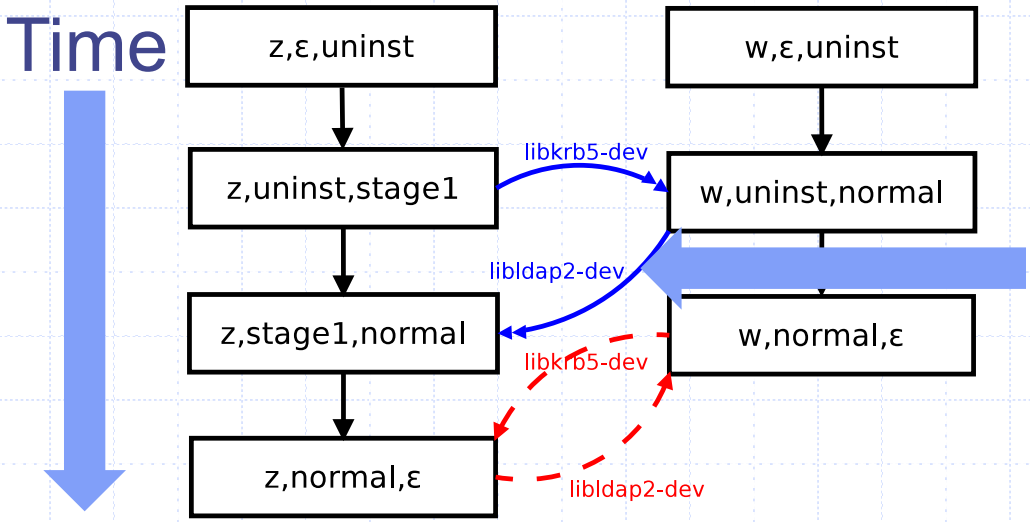
- Use the graph of the reachability algorithm **bottom-up** from the target state
  - select the **bindings** (red arrows)
  - select the **predecessors** (black arrows)



# Fully automated deployment (no capacity, no conflicts) [ICTAI13]




- Generate an **abstract plan** (one component for each maximal path)

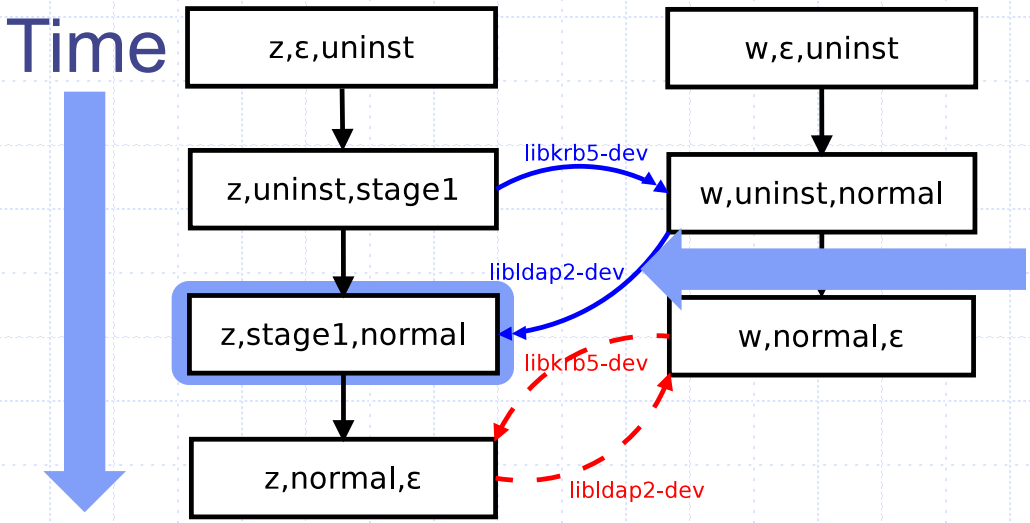



Arrows represent a **precedence** relation:
**blue**: start requirement
**red**: end requirement

# Fully automated deployment (no capacity, no conflicts) [ICTAI13]

- Plan as a **topological** visit until target:

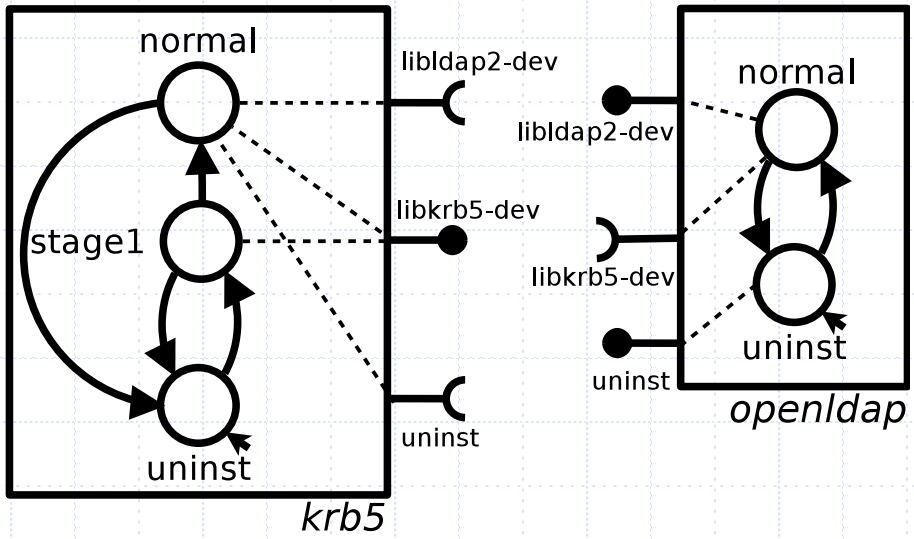
```
new(k:krb5),new(o:openldap),
stateChange(k,uninst,stage1),
bind(libkrb5-dev,o,k),stateChange(o,uninst,normal),
bind(libldap2-dev,k,o),
stateChange(k,stage1,normal)
```



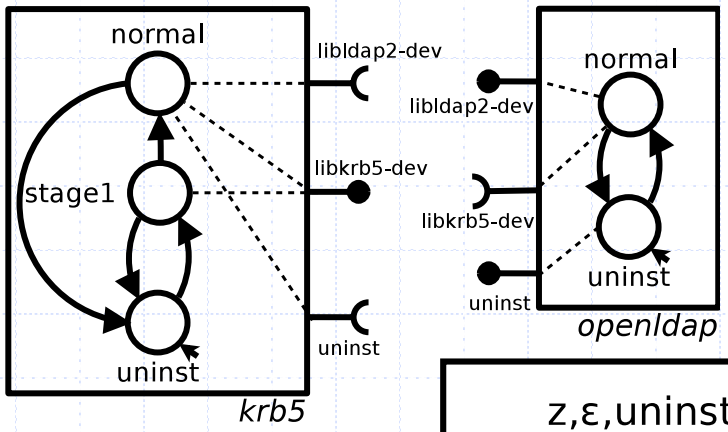

Arrows represent a **precedence** relation:
**blue**: start requirement
**red**: end requirement

# Fully automated deployment (no capacity, no conflicts) [ICTAI13]

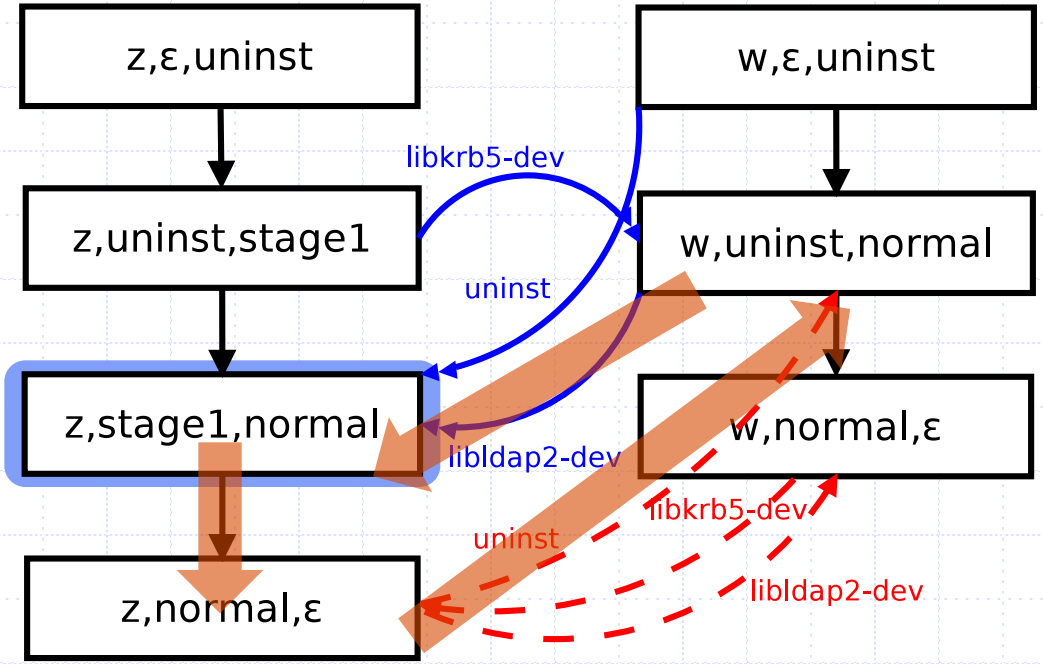
- Problem:
  cycles could forbid the **topological** visit
- Example: krb5 in normal requires an openldap in uninst state



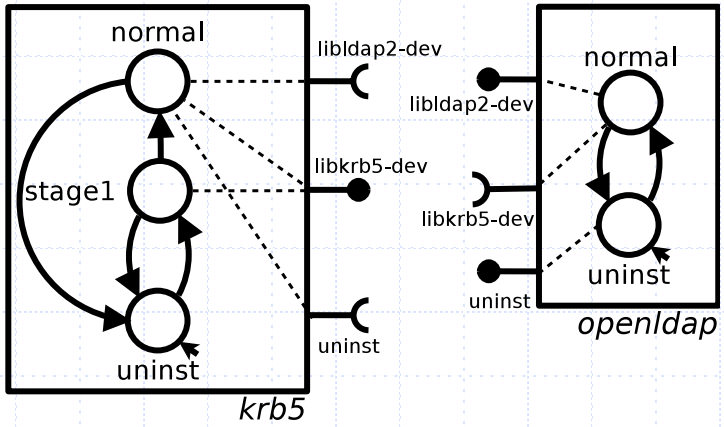
# Fully automated deployment (no capacity, no conflicts) [ICTAI13]



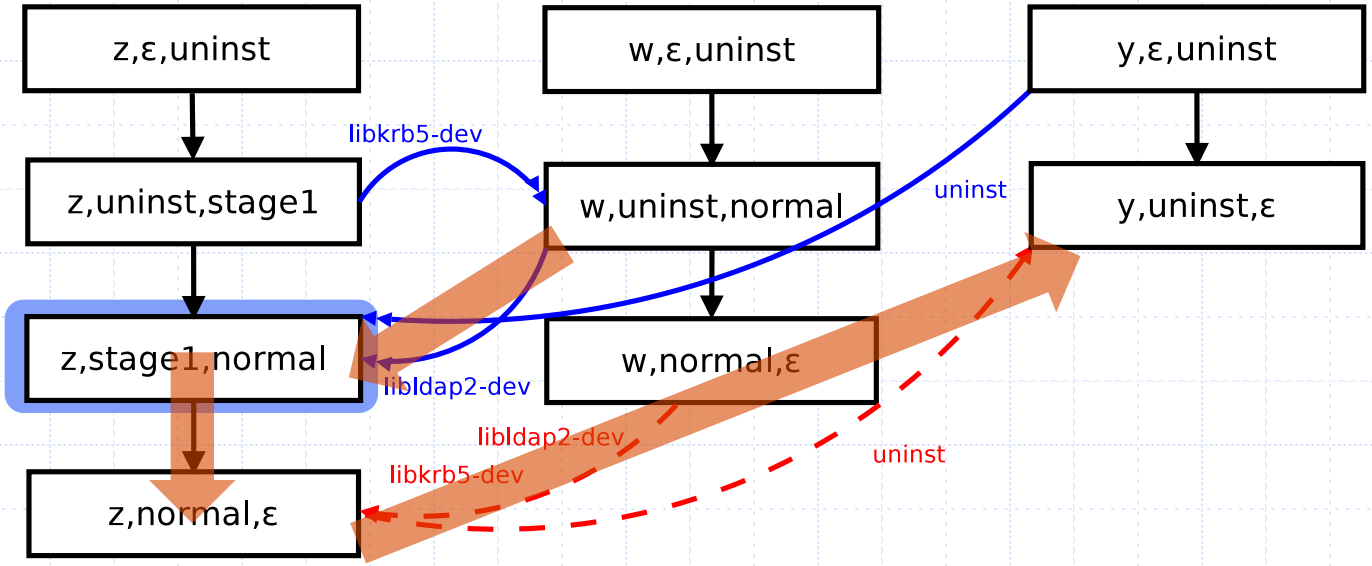

- The target state cannot be visited!



# Fully automated deployment (no capacity, no conflicts) [ICTAI13]




- Solution: component **duplication**



# Structure of the talk

- Formalizing the deployment problem
- Decidability/complexity results
- Fully automatic deployment
  (without capacity constraints and conflicts)
- **Constraints and conflicts strike back**
- Conclusion and Open issues
- Related work

# Capacity constraints and conflicts strike back [ASE14,CONCUR15]

- We have investigated the problem of synthesising the **final** configuration
  - considering **capacity constraints** and **conflicts** but…
  - …abstracting away from the internal **configuration** automata
- Idea for computing the final configuration:
  - first perform component **selection**…
  - …and then establish the **bindings**

# Component selection [CONCUR15]

- Component **selection** is NP-complete but we can use Contraint Solving technology

$$\bigwedge_{p\in\mathcal{I}} \bigwedge_{\langle\mathcal{T},q\rangle} \mathcal{T}.\mathbf{R}(q)(p) \times \mathtt{comp}(\langle\mathcal{T},q\rangle) \leq \sum_{\langle\mathcal{T}',q'\rangle} \mathtt{bind}(p,\langle\mathcal{T}',q'\rangle,\langle\mathcal{T},q\rangle)$$

$$\bigwedge_{p\in\mathcal{I}} \bigwedge_{\langle\mathcal{T},q\rangle \,.\, \mathcal{T}.\mathbf{P}(q)(p)<\infty} \mathcal{T}.\mathbf{P}(q)(p) \times \mathtt{comp}(\langle\mathcal{T},q\rangle) \geq \sum_{\langle\mathcal{T}',q'\rangle} \mathtt{bind}(p,\langle\mathcal{T},q\rangle,\langle\mathcal{T}',q'\rangle)$$

$$\bigwedge_{p\in\mathcal{I}} \bigwedge_{\langle\mathcal{T},q\rangle \,.\, \mathcal{T}.\mathbf{P}(q)(p)=\infty} \mathtt{comp}(\langle\mathcal{T},q\rangle) = 0 \;\Rightarrow\; \sum_{\langle\mathcal{T}',q'\rangle} \mathtt{bind}(p,\langle\mathcal{T},q\rangle,\langle\mathcal{T}',q'\rangle) = 0$$

$$\bigwedge_{p\in\mathcal{I}} \bigwedge_{\langle\mathcal{T},q\rangle \,.\, \mathcal{T}.\mathbf{R}(q)(p)=0 \,\wedge\, \mathcal{T}.\mathbf{P}(q)(p)>0} \mathtt{comp}(\langle\mathcal{T},q\rangle) \leq 1$$

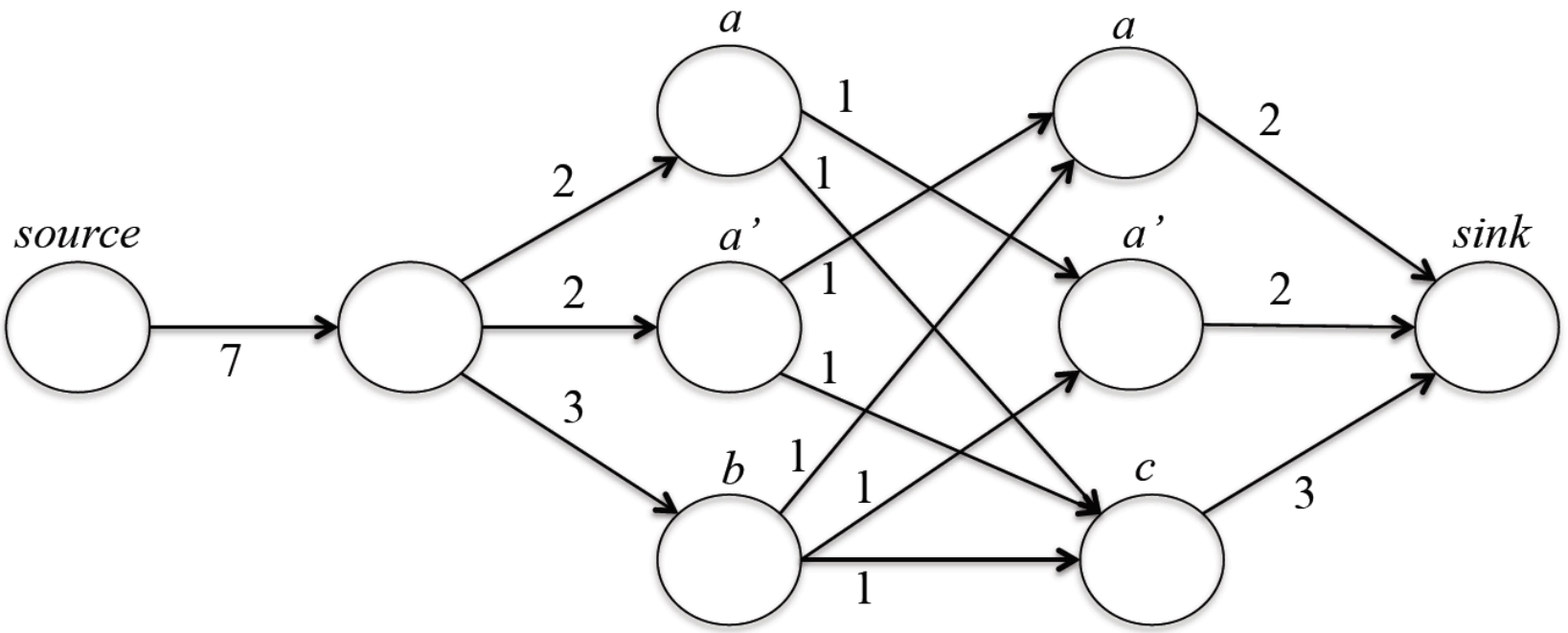
$$\bigwedge_{p\in\mathcal{I}} \bigwedge_{\langle\mathcal{T},q\rangle . \mathcal{T}.\mathbf{R}(q)(p)=0} \bigwedge_{\langle\mathcal{T}',q'\rangle\neq\langle\mathcal{T},q\rangle \,.\, \mathcal{T}'.\mathbf{P}(q')(p)>0} \mathtt{comp}(\langle\mathcal{T},q\rangle) > 0 \;\Rightarrow\; \mathtt{comp}(\langle\mathcal{T}',q'\rangle) = 0$$

$$\bigwedge_{p\in\mathcal{I}} \bigwedge_{\langle\mathcal{T},q\rangle} \bigwedge_{\langle\mathcal{T}',q'\rangle\neq\langle\mathcal{T},q\rangle} \mathtt{bind}(p,\langle\mathcal{T},q\rangle,\langle\mathcal{T}',q'\rangle) \leq \mathtt{comp}(\langle\mathcal{T},q\rangle) \times \mathtt{comp}(\langle\mathcal{T}',q'\rangle)$$

$$\bigwedge_{p\in\mathcal{I}} \bigwedge_{\langle\mathcal{T},q\rangle} \mathtt{bind}(p,\langle\mathcal{T},q\rangle,\langle\mathcal{T},q\rangle) \leq \mathtt{comp}(\langle\mathcal{T},q\rangle) \times (\mathtt{comp}(\langle\mathcal{T},q\rangle) - 1)$$

# Bindings establishment [CONCUR15]

- **Bindings** decided as solution of a max-flow problem

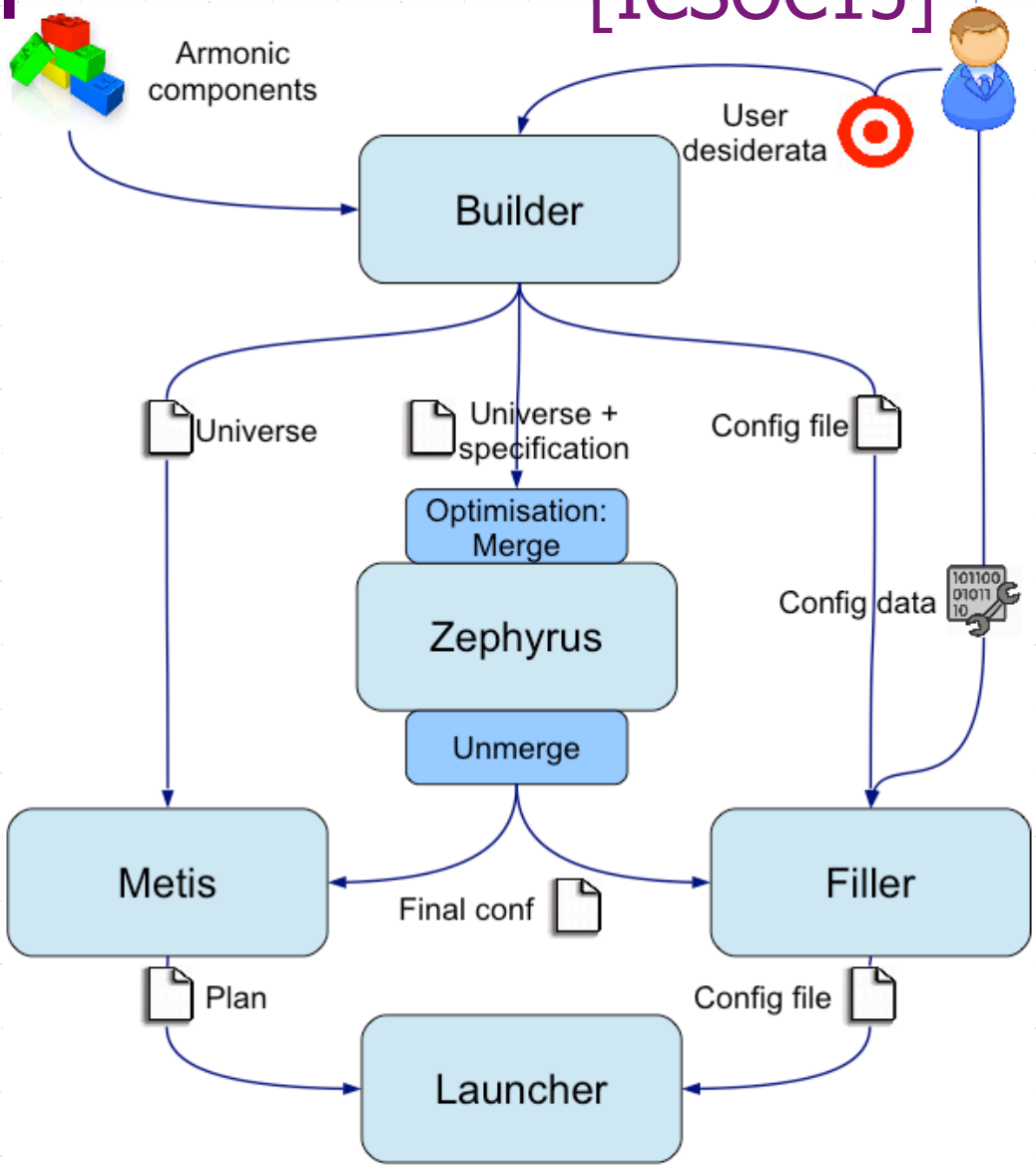


# Structure of the talk

- Formalizing the deployment problem
- Decidability/complexity results
- Fully automatic deployment
  (without capacity constraints and conflicts)
- Constraints and conflicts strike back
- **Conclusion and Open issues**
- Related work

# Putting everything together: Aeolus Blender [ICSOC15]

- **Armonic**: library of components
- **Zephyrus**: synthesis of the final architecture
- **Metis**: plan the configuration actions



# Reconfiguration vs. Deployment

- **Reconfiguration** problem:
  - same as deployment,
    but with **non empty** initial configuration
  - reconfiguration is already **PSpace**-complete without capacity constraints and conflicts

    [MFCS15]
- **Open issue**:
  - Find further **restrictions** to the model that make reconfiguration tractable
    (seems very useful in practice)

# Other open issues

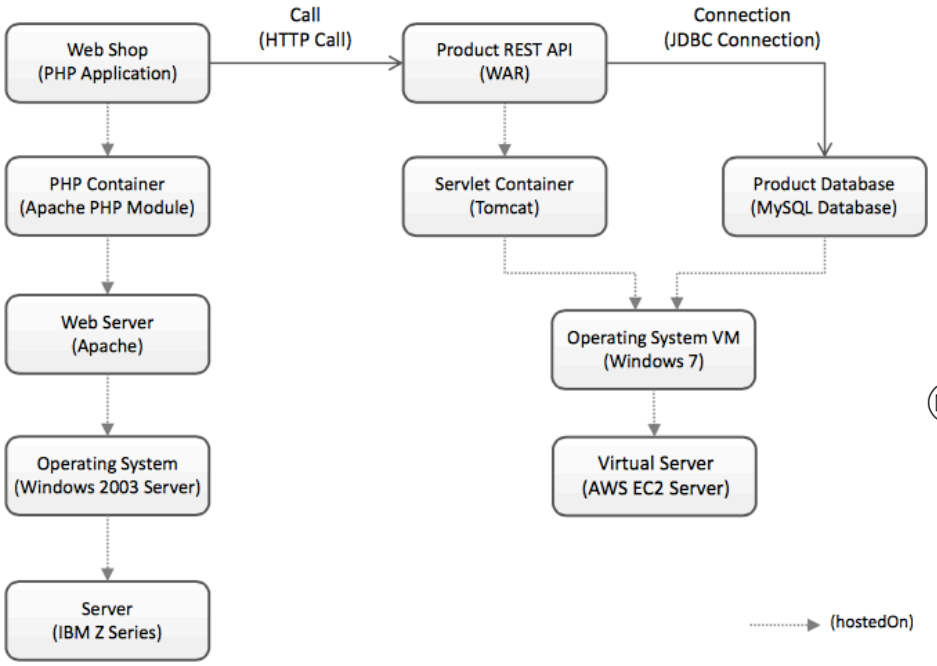
- In real systems there is a flow of configuration data among components:
  - Room for **name-passing** models?
- Hierarchical modeling (administrative domains, cloud providers, geographical areas, …):
  - Room for **ambient-like** models?
- QoS and resource consumption:
  - Room for **quantitative** models?

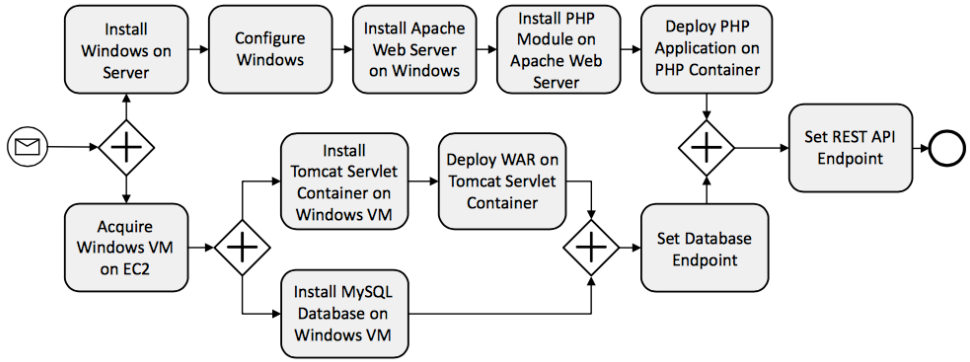
# Structure of the talk

- Formalizing the deployment problem
- Decidability/complexity results
- Fully automatic deployment
  (without capacity constraints and conflicts)
- Constraints and conflicts strike back
- Conclusion and Open issues
- **Related work**

# Related work

- TOSCA [OASIS standard 2013]
  - Language for topologies and deployment plans

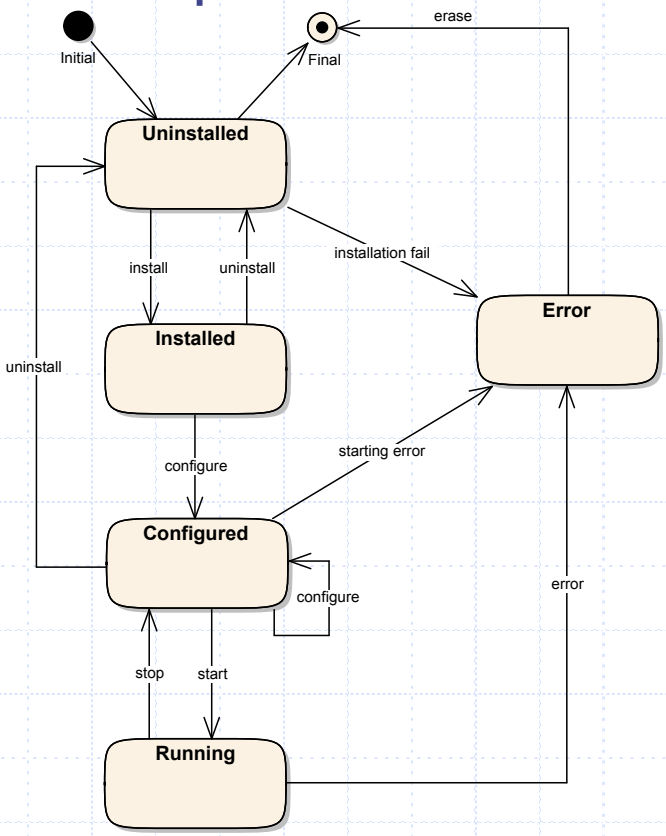


# Related work

- CloudMF [J.Ferry et al. - NordiCloud13]
  - Similar language for component description

```
{
  "id" : "SensApp",
  "retrieval" : "wget -P ~ http://cloudml.org/SensApp.
       war; wget -P ~ http://cloudml.org/startsensapp.
       sh ; wget -P ~ http://cloudml.org/deploysensapp.
       sh",
  "deployment" : "deploysensapp.sh",
  "start" : "startsensapp.sh",
  "requires" : [
    { "id" : "JettyCapability", "isOptional" : false },
    { "id" : "MongoDBCapability", "isOptional" : false
        }
  ],
  "inputs" : [
    {
      "id" : "RESTChannel",
      "portNumber" : "8080",
      "isRemote" : true
    }
  ],
  "provides" : [
    {
      "id" : "RESTServer",
      "portNumber" : "8080"
    }
  ]
}
```



# Related work

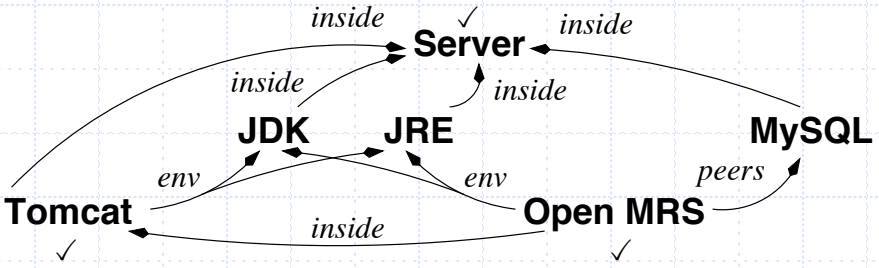
- ConfSolve [J.A.Hewson, P.Anderson, A.D.Gordon - LISA12]
  - **Object-oriented language** for services and machines
  - **Type system** for checking configuration correctness
  - **Constraint solver** for automatic placement of services on machines

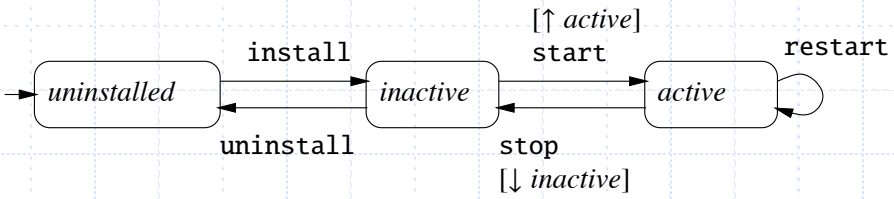
```
class Machine {
  var os as OperatingSystem;
  var cpus as 1..4;
  var memory as int;
}
```

```
class DatabaseServer extends Role {
  var role as DatabaseRole;

  // slave or master
  var peer as ref DatabaseServer;

  // the peer cannot be itself
  peer != this;

  // a master's peer must be a slave,
  // and a slave's peer must be a master
  role != peer.role;
}
```

# Related work

- Engage [J.Fischer, R.Majumdar, S.Esmaeilsabzali - PLDI12]
  - **Architectural specification** in terms of inside / peer / environment relationships
  - Automata with resource **lifecycle** and transient **dependencies**
  - Assumption on **acyclic** relationships (to always guarantee topological visit)



# Mentioned publications


- [SEFM12] R. Di Cosmo, S. Zacchiroli, G. Zavattaro.
  *Towards a Formal Component Model for the Cloud.*
  Proc. of SEFM'12: 156-171. LNCS 7504, Springer.
- [ICALP13] R. Di Cosmo, J. Mauro, S. Zacchiroli, G. Zavattaro.
  *Component Reconfiguration in the Presence of Conflicts.*
  Proc. of ICALP'13: 187-198. LNCS 7966, Springer.
- [ICTAI13] T. A. Lascu, J. Mauro, G. Zavattaro.
  *A Planning Tool Supporting the Deployment of Cloud Applications.*
  Proc. of ICTAI'13: 213-220. IEEE Press.
- [I&C14] R. Di Cosmo, J. Mauro, S. Zacchiroli, G. Zavattaro.
  *Aeolus: A component model for the cloud.*
  Information and Computation, 239: 100-121 (2014).
- [MFCS15] J. Mauro, G. Zavattaro.
  *On the Complexity of Reconfiguration in Systems with Legacy Components.*
  Proc. of MFCS'15: 382-393. LNCS 9234, Springer.
- [ASE14] R. Di Cosmo, M. Lienhardt, R. Treinen, S. Zacchiroli, J. Zwolakowski, A. Eiche, A. Agahi.
  Automated synthesis and deployment of cloud applications.
  Proc. of ASE'14: 211-222. ACM Press.
- [ICSOC15] R. Di Cosmo, A. Eiche, J. Mauro, S. Zacchiroli, G. Zavattaro, J. Zwolakowski.
  *Automatic Deployment of Services in the Cloud with Aeolus Blender.*
  Proc. of ICSOC'15, to appear, Springer.