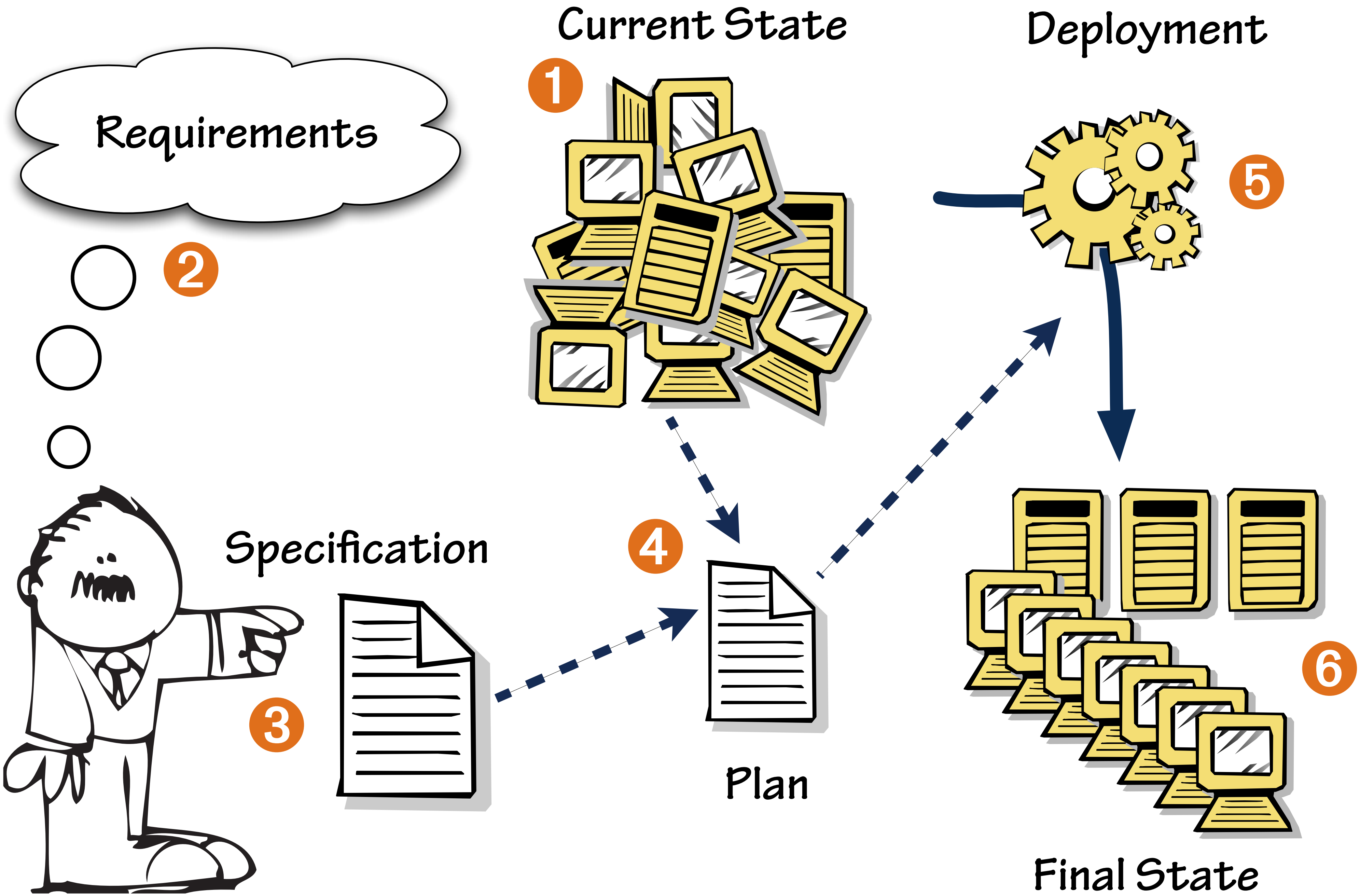
Constraints, Agents & Planning in System Configuration

Paul Anderson
<dcspaul@ed.ac.uk>

http://homepages.inf.ed.ac.uk/dcspaul/publications/pepa-2012.pdf

THE UNIVERSITY *of* EDINBURGH
informatics

cisa | Centre for Intelligent Systems and their Applications


Requirements
Current State
Deployment
1
2
5
Specification
4
3
Plan
6
Final State

➊➋➌

Three Projects

- Constraint-based specifications
  - how do we turn our “common sense” requirements” into a concrete specification that can be implemented automatically?
- Agent-based configuration
  - how can we decentralise some configuration decisions, but retain an overall control of the policy?
- Planning for configuration change
  - how do we create a sequence of operations which will transform “what we have” into “what we want” without breaking anything in the process?

Constraint-Based Specifications

With John Hewson
<john.hewson@ed.ac.uk>

http://homepages.inf.ed.ac.uk/s0968244/

Sponsored by Microsoft Research

Constraint-Based Specifications

- At some point all the details of the final configuration need to be worked out
- But specifying these all explicitly is not a good idea
  - overspecification allows no room for autonomic adjustment (except by non-declaratiave rules)
  - fully-instantiated configurations are hard to compose with other people's requirements
  - it is hard and mistakes are likely
- We want to specify the minimum necessary to meet our requirements
  - and leave the system the freedom to fill in the details

ConfSolve

- Confsolve is a declarative configuration language
  - we can specify the structure of the final configuration
  - not the procedures necessary to achieve it
- ConfSolve allows us specify "loose" configurations
  - we can specify some constraints on the final values without giving explicit values
- ConfSolve uses a standard constraint solver to generate a concrete configuration
- The output can be transformed into "Puppet" or some other standard configuration language for deployment

Some ConfSolve Classes

```
class Service {
    var host as ref Machine
}
class Datacenter {
    var machines as Machine[8]
}

class Machine { }

class Web_Srv extends Service { }

class Worker_Srv extends Service { }

class DHCP_Srv extends Service { }
```

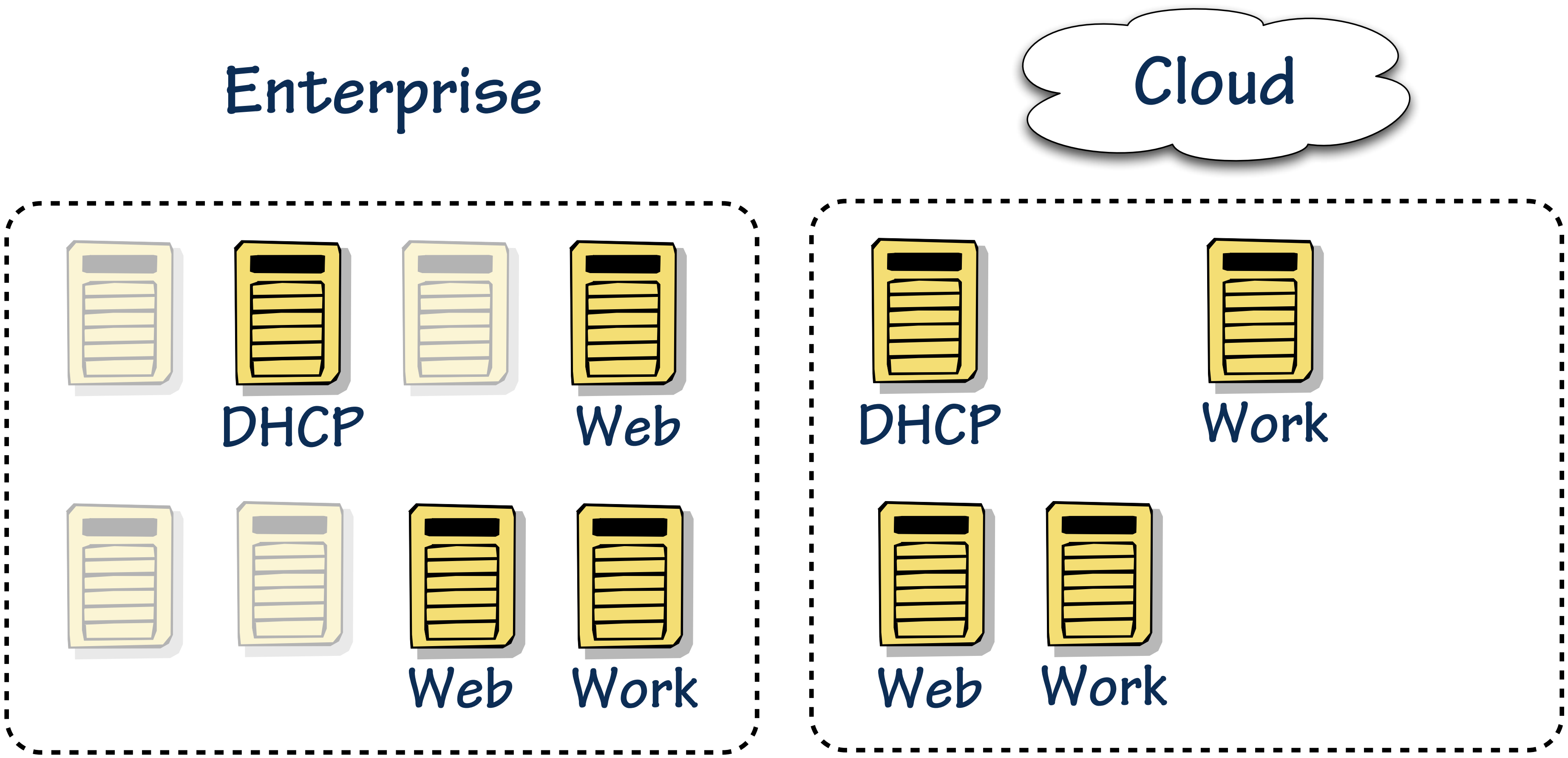
Two Datacentres
Three Services

```
var cloud as Datacenter
var enterprise as Datacenter

var dhcp as DHCP_Service[2]
var worker as Worker_Service[3]
var web as Web_Service[3]
```

No Two Services on the Same Machine

```
var services as ref Service[7]

where foreach (s1 in services) {
    foreach (s2 in services) {
        if (s1 != s2) {
            s1.host != s2.host
        }
    }
}
```

Constraint Solution



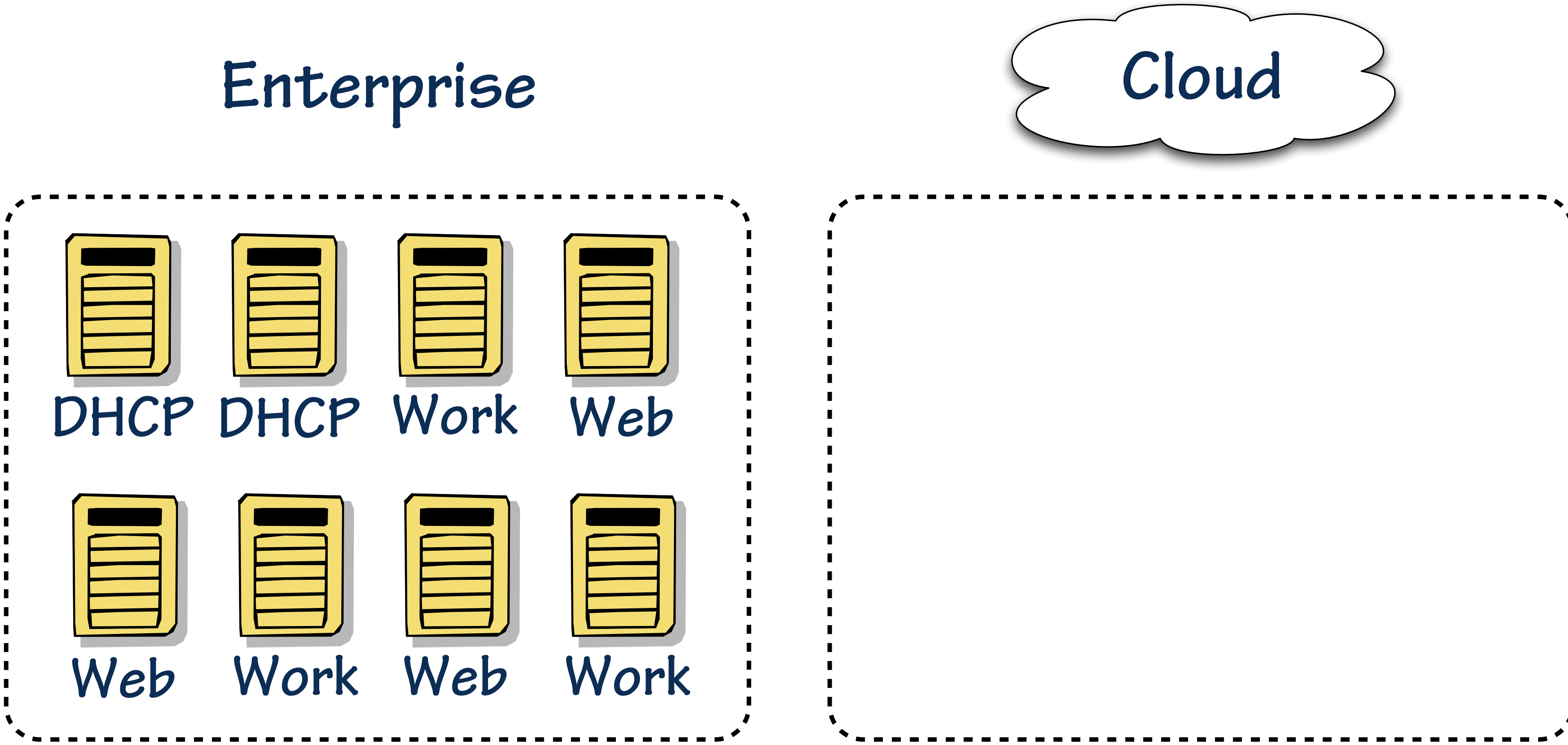
Not a good solution!

The constraints are too loose

Favour Placement of Machines in the Enterprise

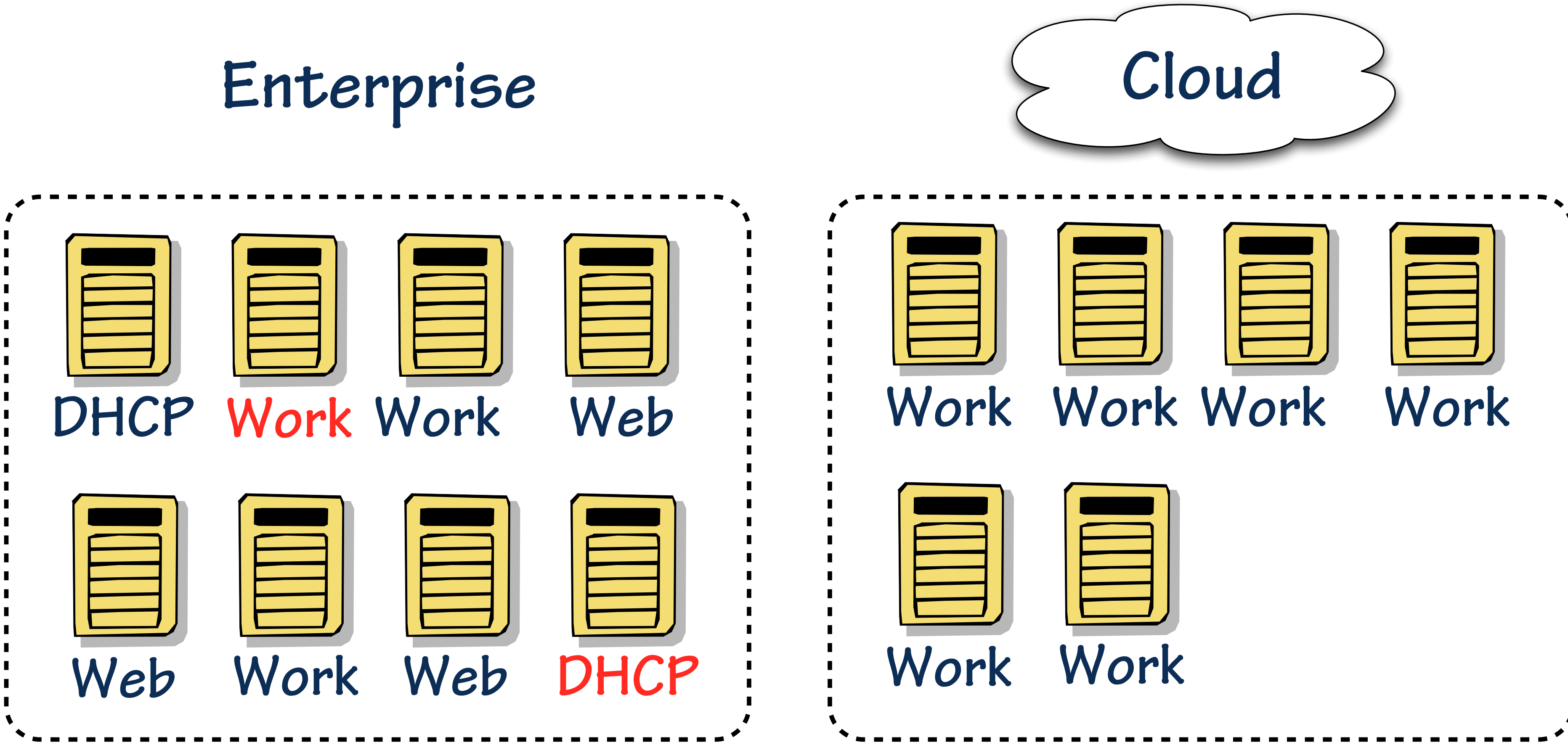
```
var utilisation as int

where utilisation == count (
  s in services
  where s.host in enterprise.machines)

maximize utilisation
```

Constraint Solution



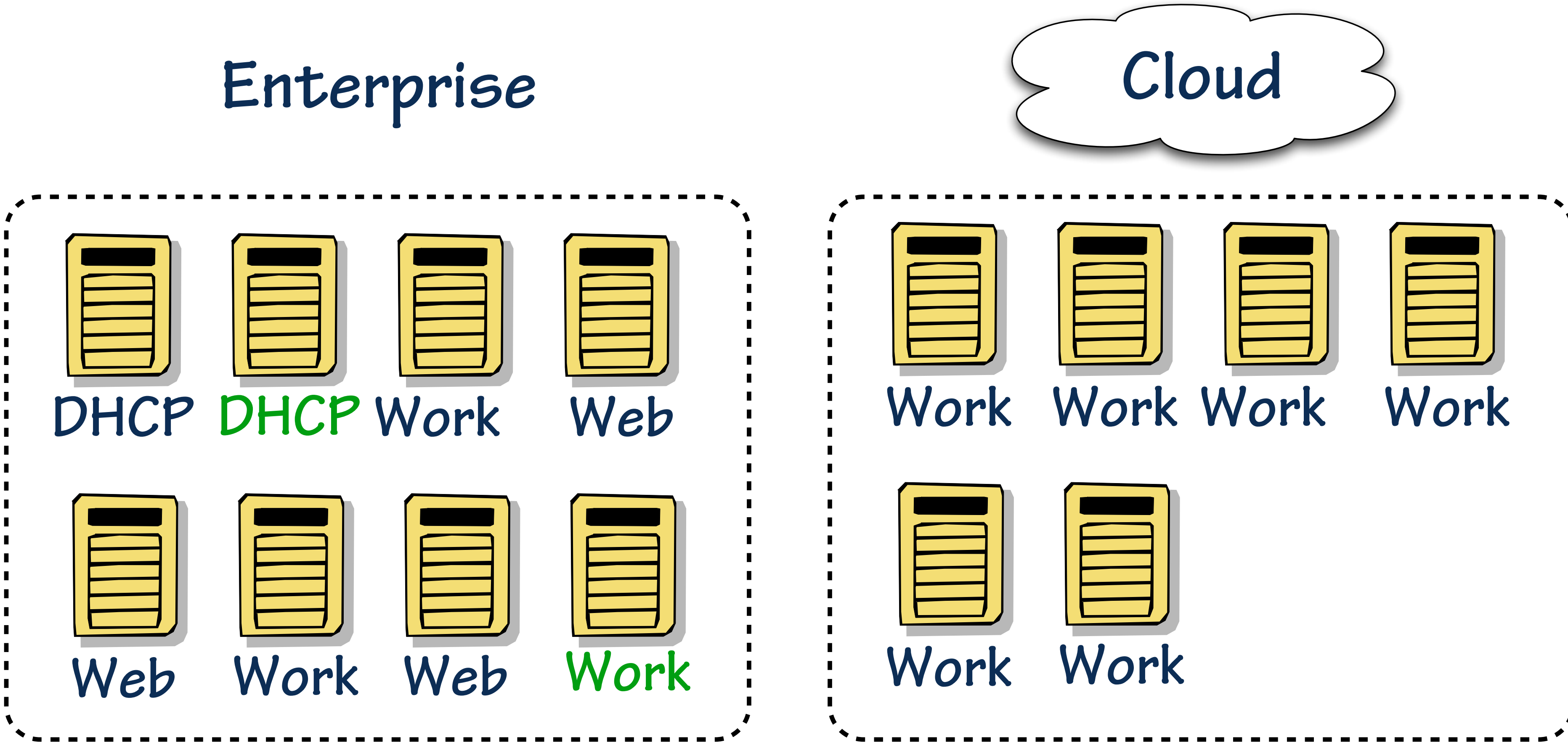
A much better solution

Add Six More Workers



The new solution results in a different allocation for the enterprise which causes an unwanted migration

“Minimal Change” Constraints



If we add constraints to minimise the “distance” from the old solution, we introduce some “stability”

Some Issues

- We would like the optimisation function to take account of user preferences as well:
  - "put these two servers on the same network IF POSSIBLE"
- This is easy to do, but:
  - how do we weight the priorities for all the different preferences to always get a sensible outcome?
  - is it more important to keep these servers on the same network, or to maintain the stability?
- We can express all of these things, but we want to do so in a way which makes sense to the user and is not so complicated as to be unpredictable

❶❷❸

Agent-Based Configuration

Work with Shahriar Bijani
<S.Bijani@sms.ed.ac.uk>

http://homepages.inf.ed.ac.uk/s0880557

Centralised Configuration?

- Centralised configuration
  - allows a global view with complete knowledge
- But ...
  - it is not scalable
  - it is not robust against communication failures
  - federated environments have no obvious centre
  - different security policies may apply to different subsystems
- The challenge ...
  - devolve control to an appropriately low level
  - but allow high-level policies to determine the behaviour

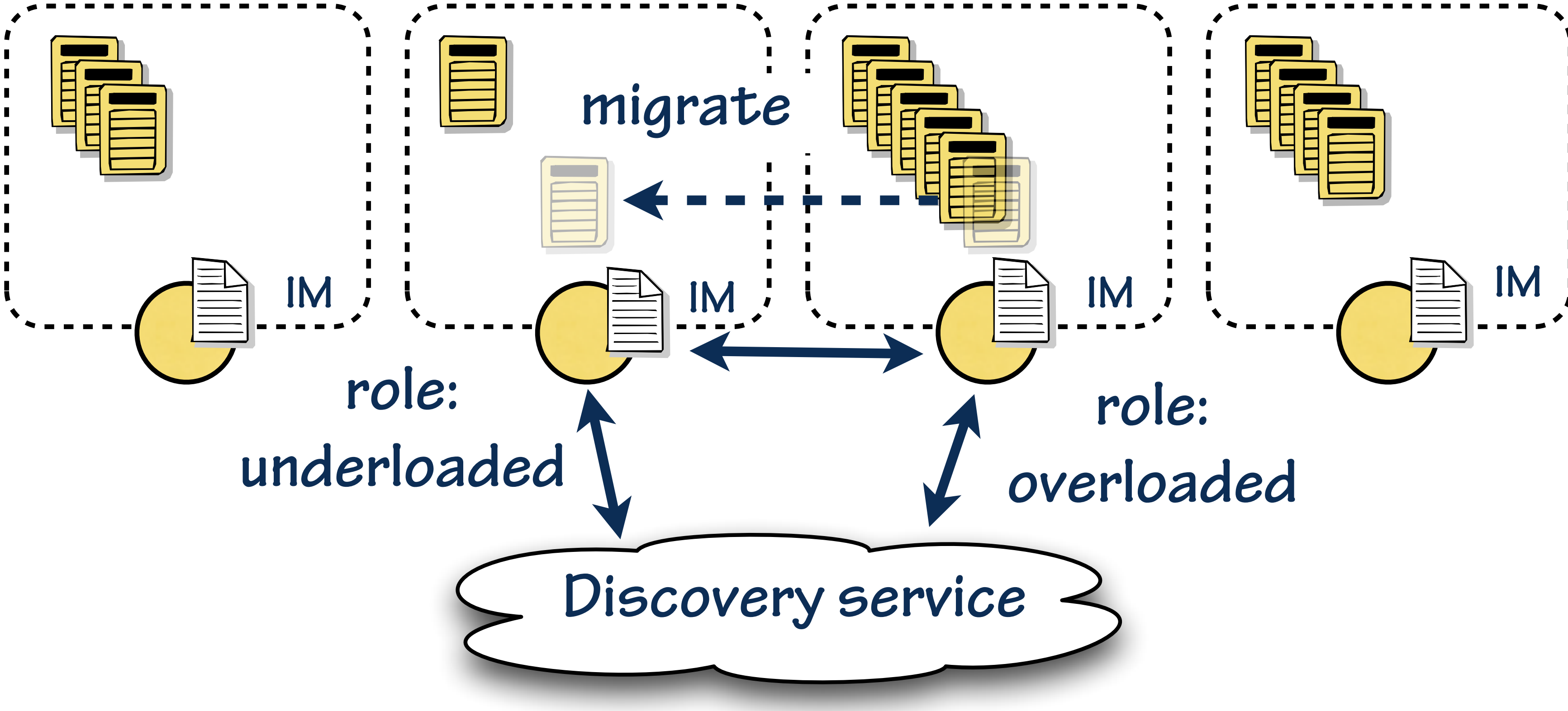
“OpenKnowledge” & LCC

- Agents execute “interaction models”
- Written in a “lightweight coordination calculus” (LCC)
- This provides a very general mechanism for doing distributed configuration
- Policy is determined by the interaction models themselves which can be managed and distributed from a central point of control
- The choice of interaction model and the decision to participate in a particular “role” remains with the individual peer
  - and hence, the management authority

A Simple LCC Example

```
a(buyer, B) ::
 ask(X) => a(shopkeeper, S) then
 price(X,P) <= a(shopkeeper, S) then
 buy(X,P) => a(shopkeeper, S)
                 ← afford(X, P) then
 sold(X,P) <= a(shopkeeper, S)

a(shopkeeper, S) ::
  ask(X) <= a(buyer, B) then
  price(X, P) => a(buyer, B)
                  ← in_stock(X, P)then
  buy(X,P) <= a(buyer, B) then
  sold(X, P) => a(buyer, B)
```

An Example: VM Allocation



- Policy 1 - power saving
  - pack VMs onto the minimum number of physical machines
- Policy 2 - agility
  - maintain an even loading across the physical machines

An Idle Host

```
a(idle, ID1) ::
        null
        ← overloaded(Status)
     then
        a(overload(Status), ID1)
   ) or (
        null
        ← underloaded(Status)
     then
        a(underload(Status), ID1)
   ) or (
     a(idle, ID1)
   )
```

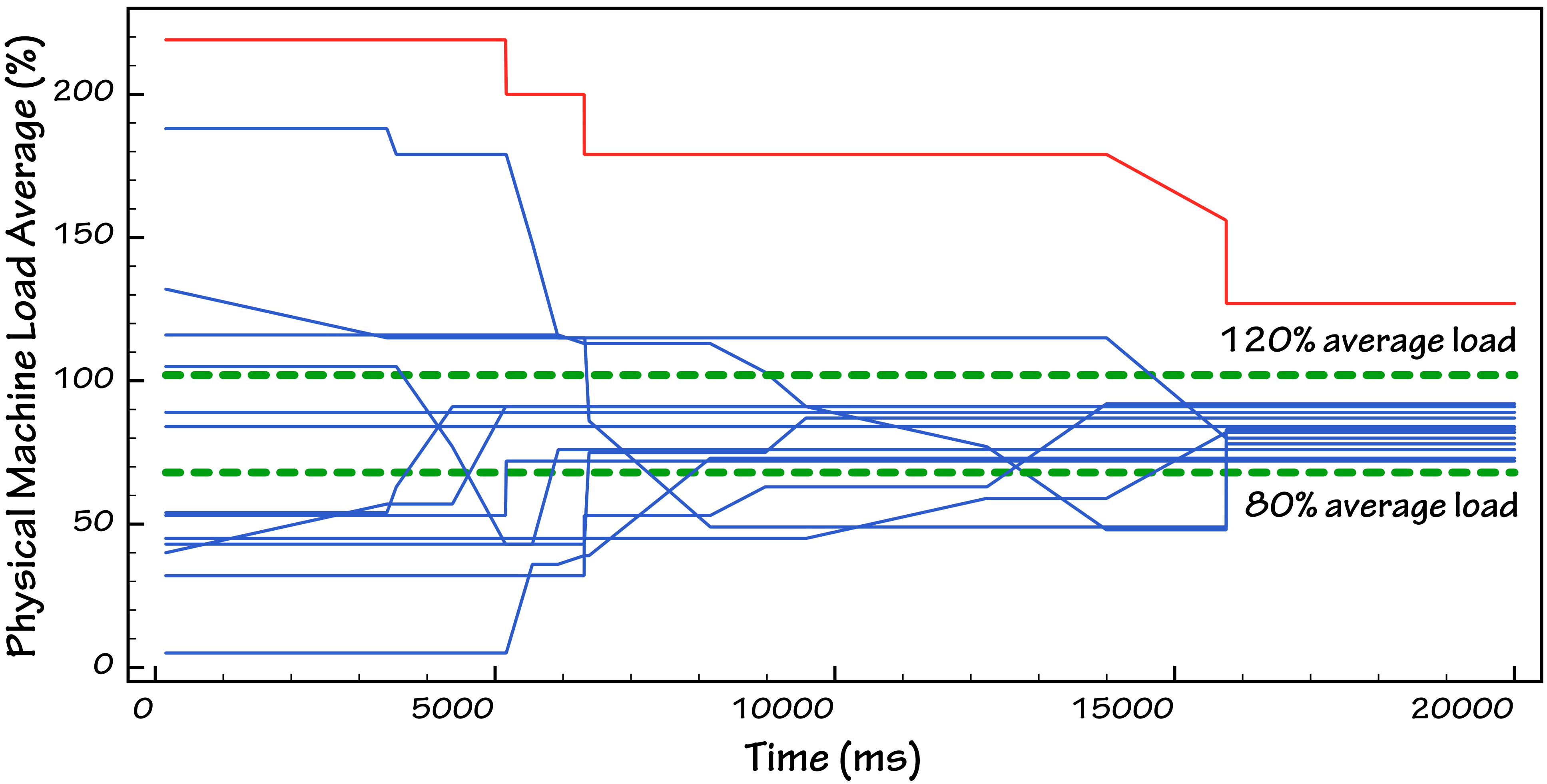
An Overloaded Host

```
a(overloaded(Need), ID2) ::
    readyToMigrate(Need)
    => a(underloaded, ID3)
  then
    migration(OK)
    <= a(underloaded, ID3)
  then
    null
    ← migration(ID2, ID3)
  then
    a(idle, ID2)
```

An Underloaded Host

```
a(underloaded(Capacity), ID3) ::
    readyToMigrate(Need)
    <= a(overloaded, ID2)
  then
    migration(OK)
    => a(overloaded, ID2)
    ← canMigrate(Capacity, Need)
  then
    null ← waitForMigration()
  then
    a(idle, ID3)
```

A Simulation



Some Issues

- LCC can be used to implement more sophisticated protocols - such as "auctions" which are ideal for many configuration scenarios
- But some things are hard to do without global knowledge
  - balance the system so that all the machines have exactly the same load?
- Handling errors and timeouts in an unreliable distributed system is hard

1 2 3

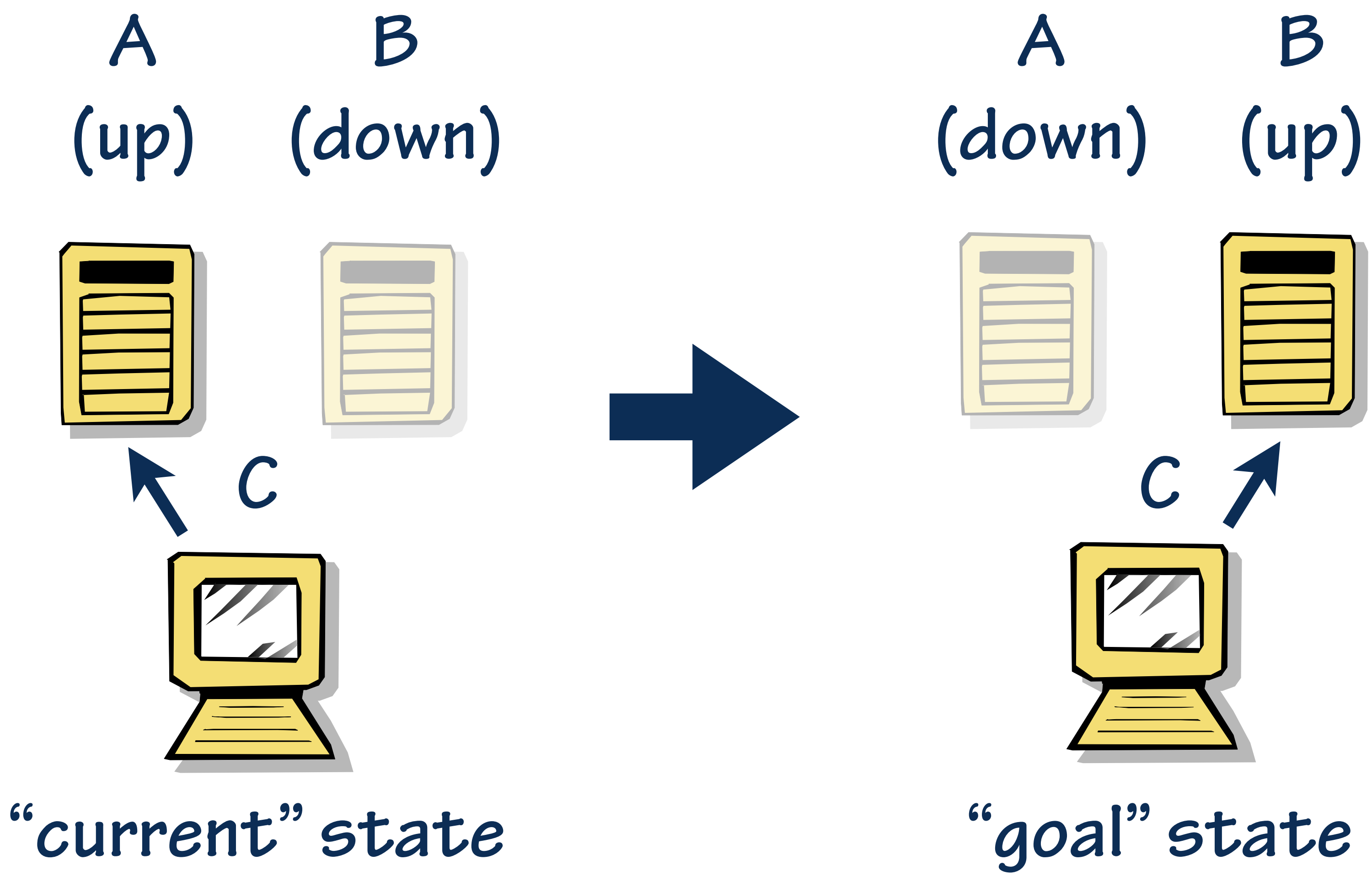
Planning for Configuration Change

Work with Herry
<H.Herry@sms.ed.ac.uk>

http://homepages.inf.ed.ac.uk/s0978621

Sponsored by HP Research

An Example Reconfiguration

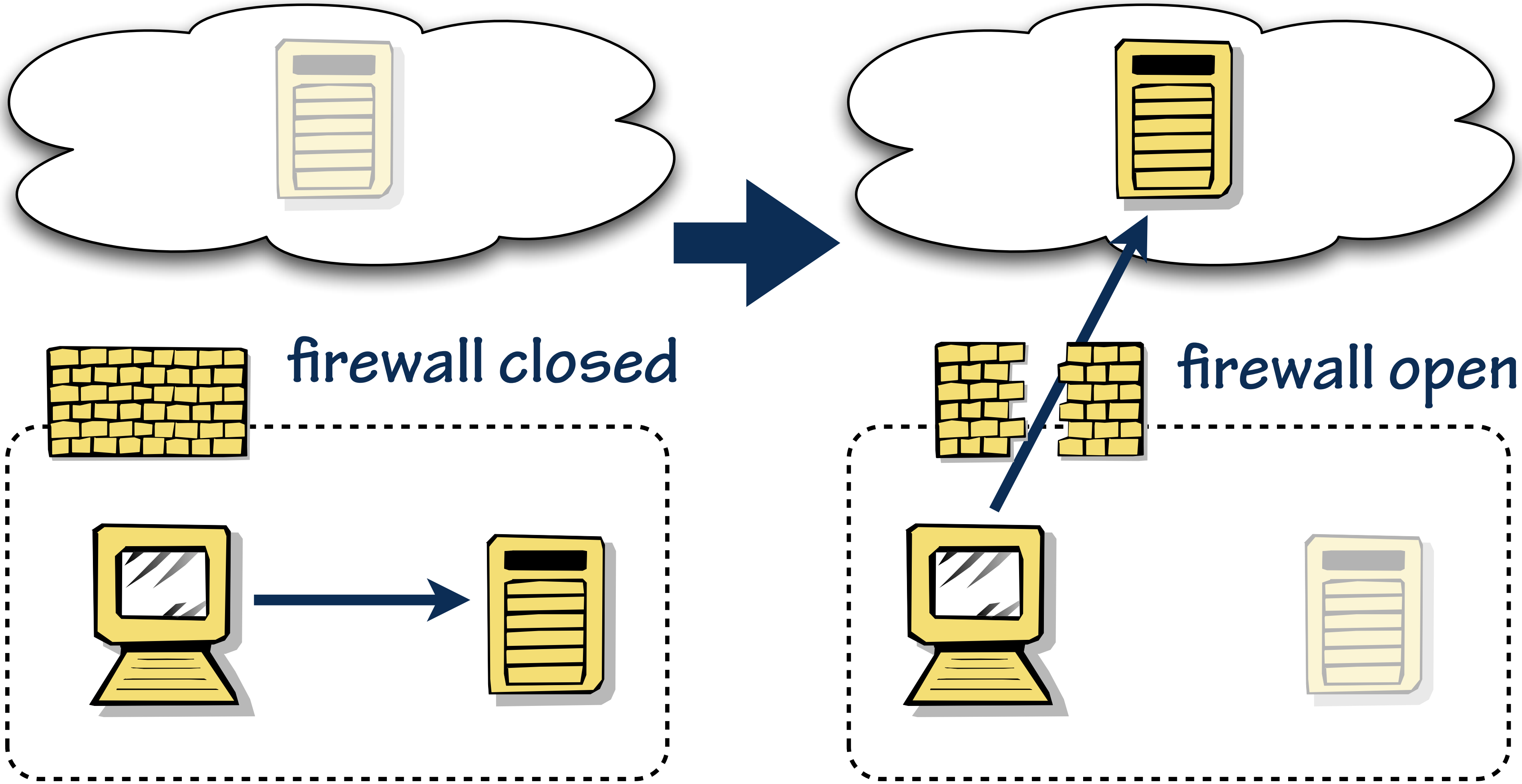


constraint: C is always attached to a server which is "up"

Possible Plans

1. A down, B up, C.server=B ✘
2. A down, C.server=B, B up ✘
3. B up, A down, C.server=B ✘
4. B up, C.server=B, A down ✔
5. C.server=B, A down, B up ✘
6. C.server=B, B up, A down ✘

"Cloudburst"

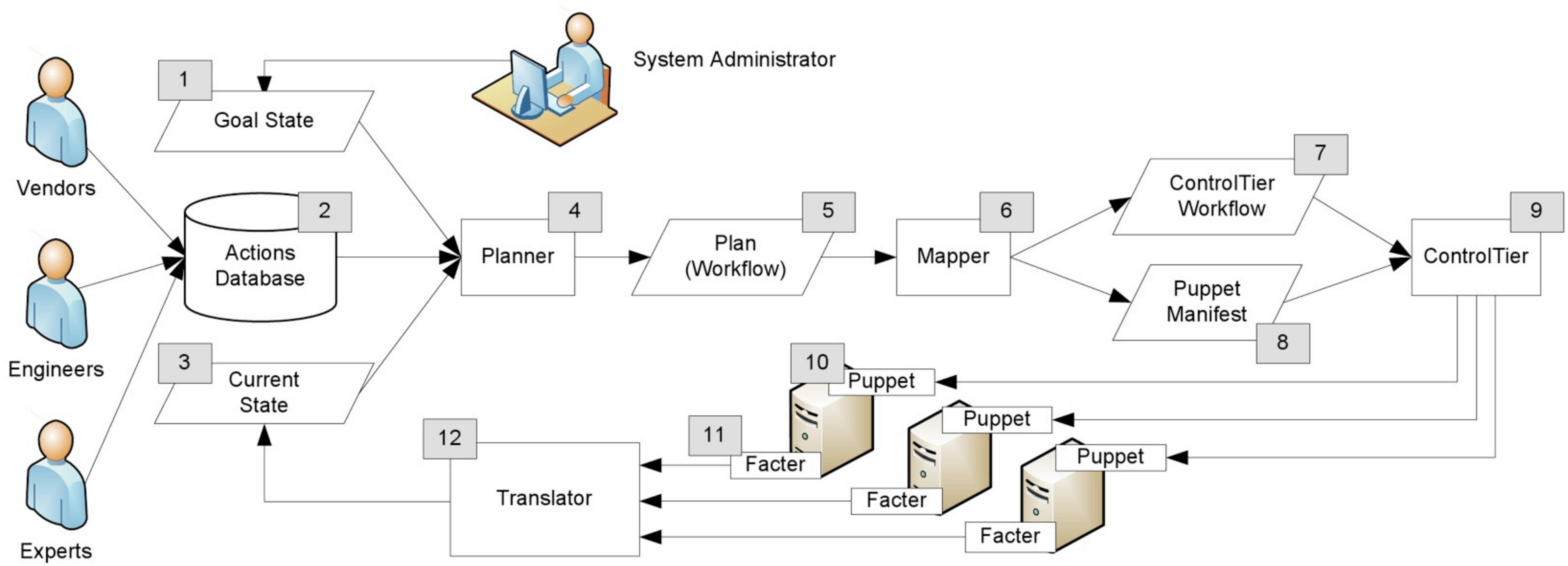


- Perhaps we need to change the DNS for the server ...
- Maybe the server needs to access internal services ...

Automated Planning

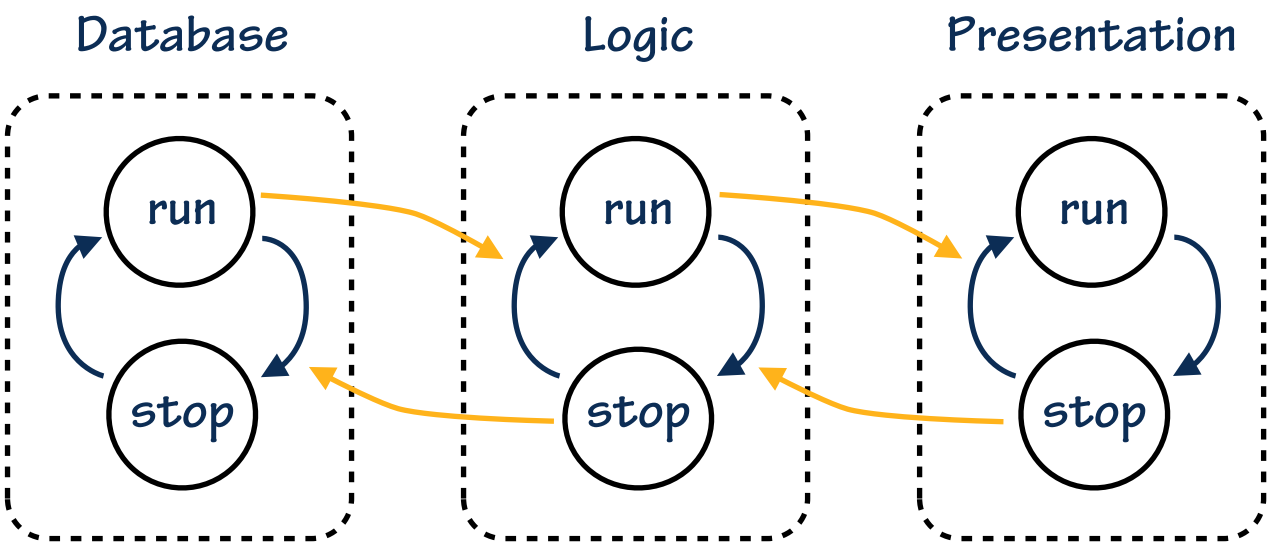
- Fixed plans ("workflows") cannot cover every eventuality
- We need to prove that any manual plans
  - always reach the desired goal state
  - preserve the necessary constraints during the workflow
- The environment is a constant state of flux
  - how can we be sure that the stored plans remain correct when the environment has changed?
- Automated planning solves these problems

A Prototype



- Current state and goal state input to planner together with a database of possible actions
- Planner (LPG) creates workflow
- Plan implemented with “Controltier” & “Puppet”

Behavioural Signatures



- Blue transitions are only enabled when the connected component is in the appropriate state
  - simple plans execute autonomously
- The plan executes in a distributed way
- The components are currently connected manually
  - and the behaviour needs to be proven correct in all cases

Planning with BSigs
(Herry's current Phd work)

- If we have ...
  - a set of components whose behaviour is described by behavioural signatures
  - a "current" and a "goal" state
- We can use an automated planner to generate a network of components to execute a plan which will transition between the required states
- Some interesting possibilities
  - perhaps we can use LCC agents instead of the BSigs
  - this provides more "intelligence" in the components

Constraints, Agents & Planning in System Configuration

Paul Anderson

<dcspaul@ed.ac.uk>

http://homepages.inf.ed.ac.uk/dcspaul/publications/pepa-2012.pdf

THE UNIVERSITY *of* EDINBURGH
informatics

cisa | Centre for Intelligent Systems and their Applications