Rhizoma: A Runtime for Self-deploying, Self-managing Overlays

Qin Yin*, Adrian Schüpbach*, Justin Cappos+, Andrew Baumann*, Timothy Roscoe*

* Systems Group, Department of Computer Science, ETH Zurich
+ Dept. of Computer Science and Engineering, University of Washington

Cloud computing infrastructures are emerging

amazon web services EC2

CPU	1	4	8
Memory	1.7G	7.5G	15G
Disk	160G	850G	1690G
Unit	1	2	2
Price	$0.125	$0.50	$1.00

amazon web services S3

Storage	$0.150 per GB
Transfer	$0.100 per GB
Requests	$0.01 per 10,000 GET

Deploying an online service

Challenge 1:

Selecting appropriate providers and resources

Service deployed with one provider

Service gains popularity

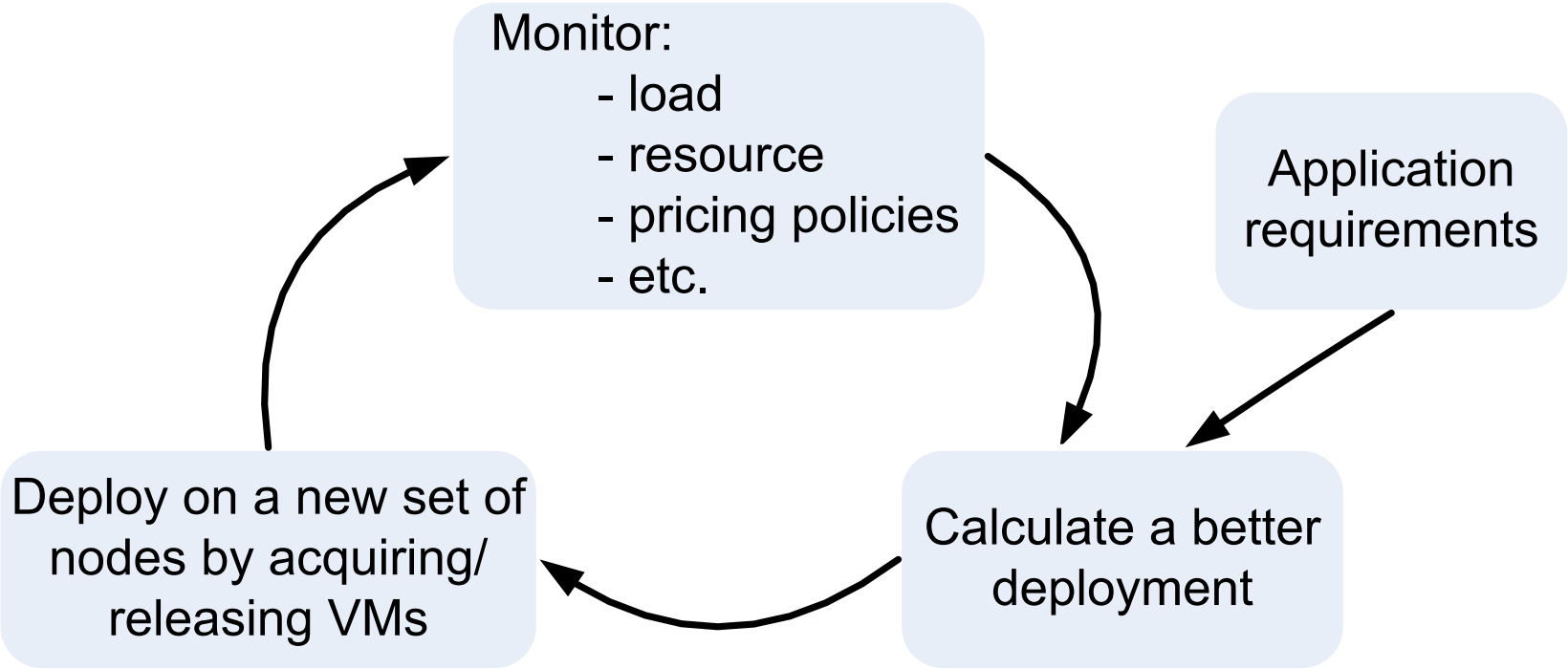
Provider outage

Problem summary

- Challenges
  - Changing resource availability
  - Changing application requirements
  - Changing pricing policies
- Current solutions
  - Human-in-the-loop
  - Separate management console (e.g. Rightscale)

Alternative approach: bundle management

- Management logic integrated with application instances
  - Instances use P2P-style self-organization
- Application "deploys itself"



Related work

- We build on existing work
  - Early PARC “worm” programs (1982)
  - Intelligent and mobile agents (1990s)
  - Autonomic computing (2000-)
- Apply the ideas to cloud environments…
  - Span multiple providers and configurations
  - Application controls resource allocation

Example application requirements (PsEPR)

- Planetary Scale Event Propagation and Router (PsEPR)
  - A unified reporting and event service for PlanetLab

P.Brett et al., “A Shared Global Event Propagation System to Enable Next Generation Distributed Services”, WORLDS’04.

- Requirements
  - 6-10 well-connected nodes
  - Lightly-loaded
  - Geographically distributed

  - Maximize CPU cycles
  - Minimize network diameter
  - Reduce budget and migration cost

Example application requirements (PsEPR)

- 6-10 well-connected nodes
- Lightly-loaded
- Geographically distributed

- Maximize CPU cycles
- Minimize network diameter
- Reduce budget and migration cost

Example application requirements (PsEPR)

- Constraints
  - 6-10 well-connected nodes
  - Lightly-loaded
  - Geographically distributed

  - Maximize CPU cycles
  - Minimize network diameter
  - Reduce budget and migration cost

Example application requirements (PsEPR)

- Constraints
  - 6-10 well-connected nodes
  - Lightly-loaded
  - Geographically distributed
- Objective
  - Maximize CPU cycles
  - Minimize network diameter
  - Reduce budget and migration cost

Example application requirements (PsEPR)

- Constraints
  - 6-10 well-connected nodes
  - Lightly-loaded
  - Geographically distributed
- Objective
  - Maximize CPU cycles
  - Minimize network diameter
  - Reduce budget and migration cost

- Constraint Optimization Problem

Idea: Apply constraint logic programming (CLP)

- Constraint Programming
  - Constraints
  - Objective function = Utility function - Cost function

Idea: Apply constraint logic programming (CLP)

- Constraint Programming
  - Constraints
  - Objective function = Utility function - Cost function

Utility function: value of a given configuration

Idea: Apply constraint logic programming (CLP)

- Constraint Programming
  - Constraints
  - Objective function = Utility function - Cost function

Cost function: cost of moving to a new configuration

Idea: Apply constraint logic programming (CLP)

- Constraint Programming
  - Constraints
  - Objective function = Utility function - Cost function

- Logic Programming
  - Heterogeneous resource information
  - Inference rules

PsEPR requirements in CLP

CONSTRAINTS

```
node_constraint(Host) :-
  comonnode{hostname: Host},
  alive(Host),
  light_loaded(Host),
  is_avail(Host),
  get_node_attr(Host, cpuspeed, Cpuspeed),
  get_node_attr(Host, freecpu, Freecpu),
  Cpuspeed*Freecpu/100 > 1.5.
```

```
group_constraint(NodeList) :-
  assemble_values(NodeList, location, Locs),
  length(NodeList, Len),
  Max is ((Len-1)//4)+1,
  ( for(I, 1, 4), param(Locs, Max) do
      count_element(I, Locs, Num),
      Num =< Max ).
```

```
path_constraint(LenList, Max) :−
  max(LenList, Max),
  diameterUtil(_, DiameterMax, _),
  Max < DiameterMax.
```

COST FUNCTION

```
migration_cost(Actions, MigrateCost) :-
  count_element(add, Actions, AddLen),
  count_element(remove, Actions, RmvLen),
  addCostParam(AddParam),
  removeCostParam(RmvParam),
  MigrateCost is AddParam*AddLen +
                 RmvParam*RmvLen.
```

UTILITY FUNCTION

```
util_function(NodeList, Util, Params) :-

  % Compute utility values for different node attributes
  fiveminloadUtil(LoadMin, LoadMax, LoadWeight),
  assemble_values(NodeList, fiveminload, LoadList),
  util_value("<", LoadList, LoadMin, LoadMax, LoadUtil),

  % Omitting utility values for liveslices and freecpu, the same as above

  % Utility of max distance from fixed nodes to the overlay
  findall(P, fixednode(P), Fixed),
  minlatency(MinLat),
  maxneighUtil(_, NeighMax, NeighWeight),
  get_nearest_neighbor_list(Fixed, NodeList, NeighList),
  max(NeighList, MaxDist),
  util_value("<", [MaxDist], MinLat, NeighMax, NeighUtil),

  % Utility of overlay network diameter
  diameterUtil(_, DiamMax, DiamWeight),
  util_value("<", Params, MinLat, DiamMax, DiamUtil),

  % Weighted average of the utilities above
  weighted_avg([LoadUtil, SliceUtil, CpuUtil, NeighUtil, DiamUtil],
  [LoadWeight, SliceWeight, CpuWeight, NeighWeight, DiamWeight], Util).
```

Definition of util_value:

$\mathbf{util\_value}_{<} = \text{avg}_i\ ((x_{max} - \text{bound}(x_i))/(x_{max} - x_{min}))$

$\mathbf{util\_value}_{>} = \text{avg}_i\ ((\text{bound}(x_i) - x_{min})/(x_{max} - x_{min}))$

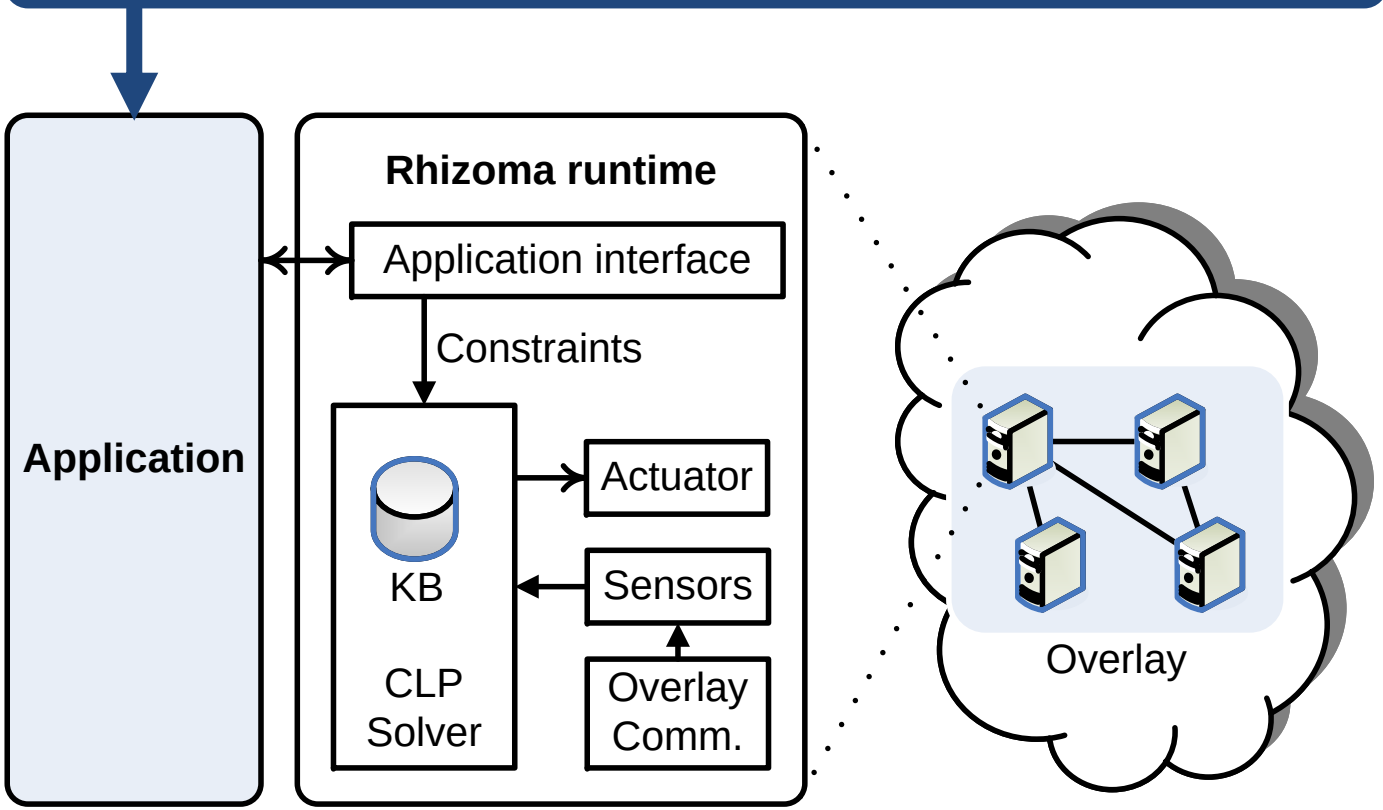
$\text{bound}(x) = \max(\min(x, x_{max}), x_{min})$

CONFIGURATIONS

fiveminloadUtil(0, 10, 2). *liveslicesUtil*(0, 10, 1). *freecpuUtil*(1, 4, 3). *maxneighUtil*(0, 500, 2). *diameterUtil*(0, 1000, 2).
addCostParam(*0.012*). *removeCostParam*(0).

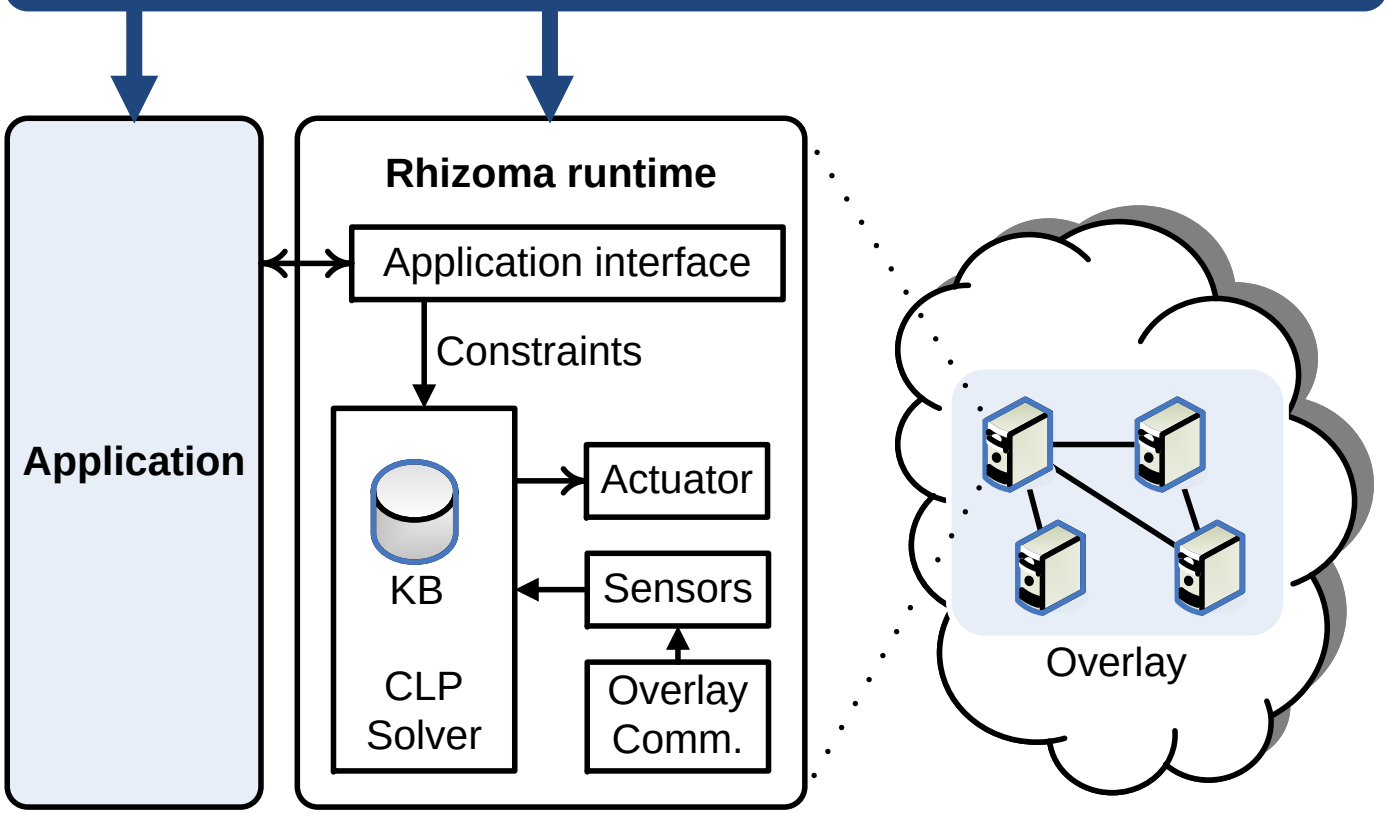
Our system: Rhizoma

- Application is a self-organizing overlay



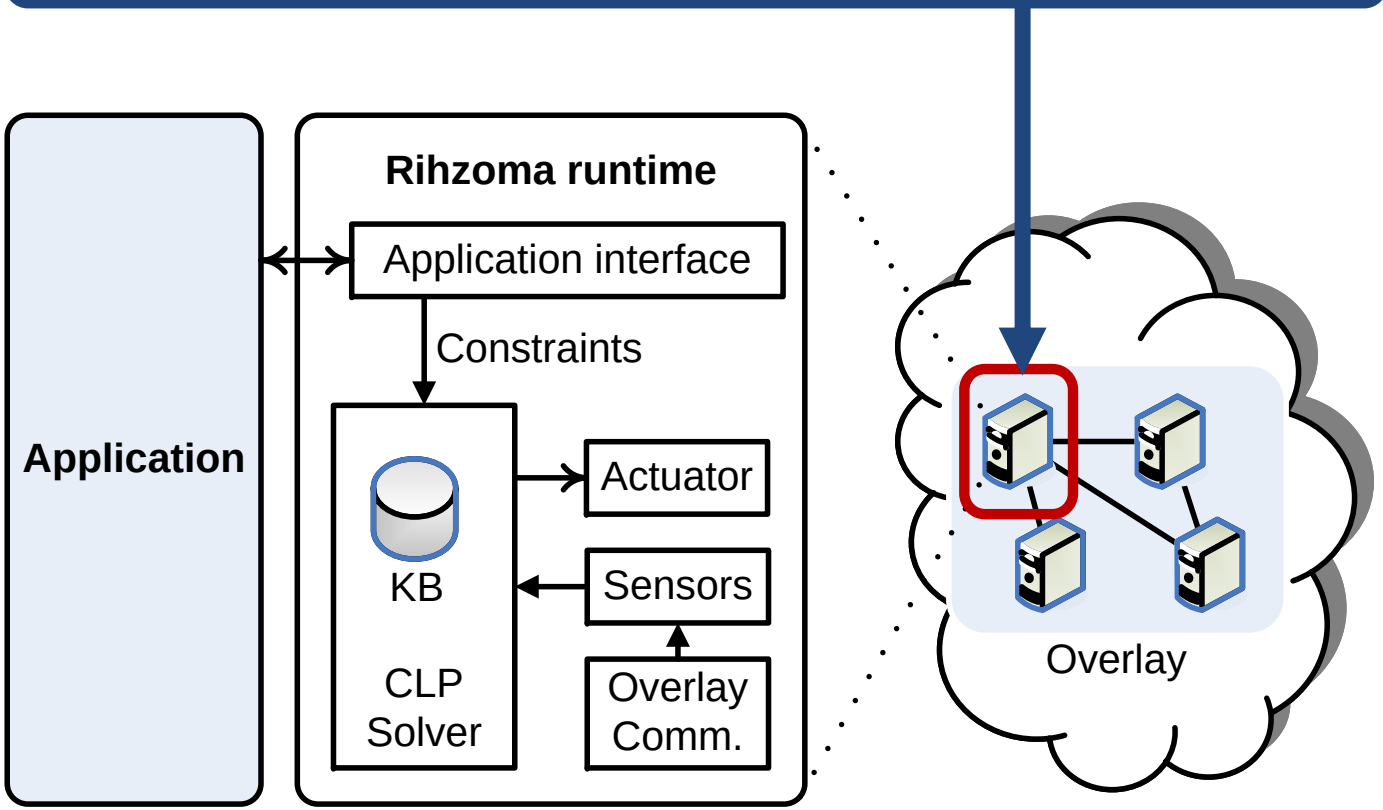
Our system: Rhizoma

- Rhizoma runtime runs alongside application



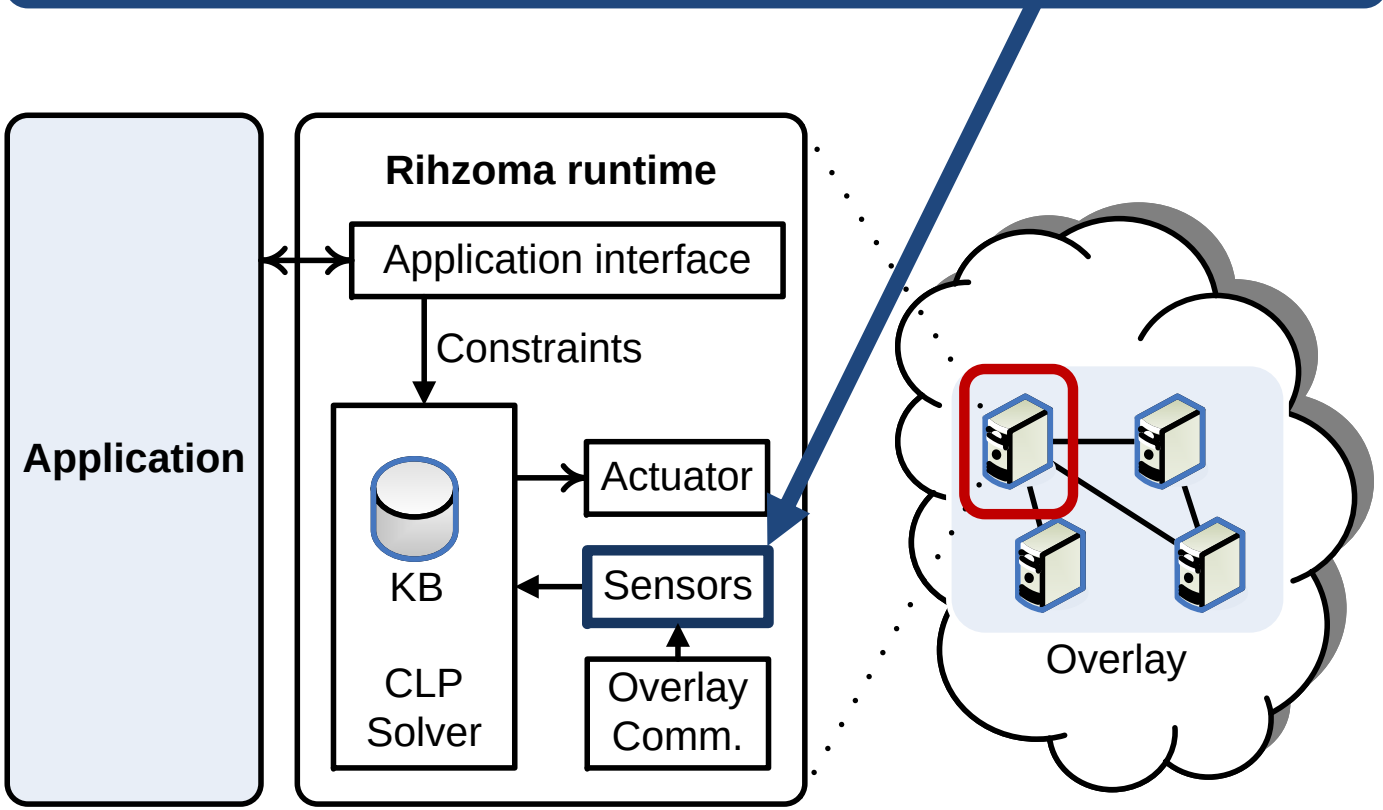
Our system: Rhizoma

- One node is elected as *coordinator* and runs CLP solver



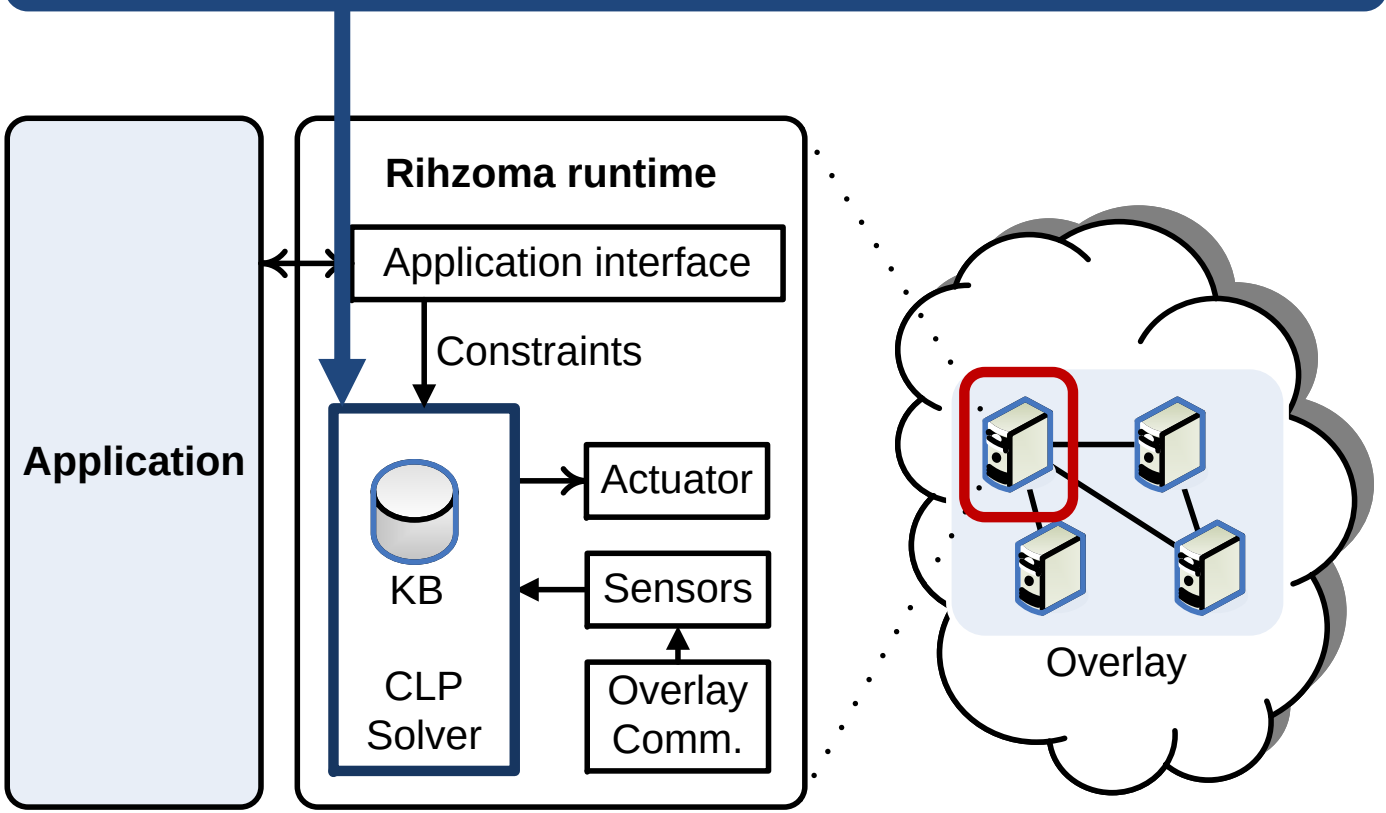
Our system: Rhizoma

- Sensors gather info from overlay and external service



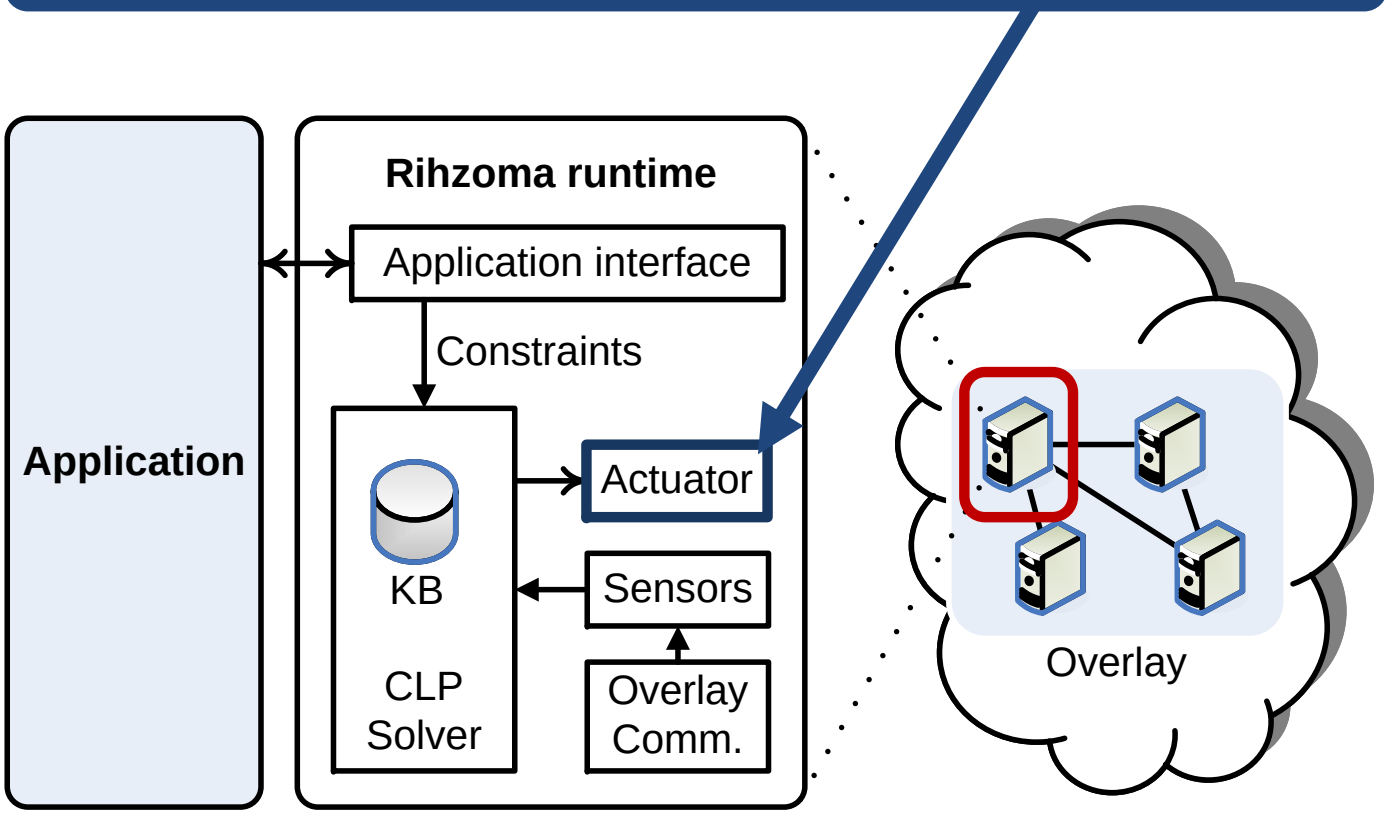
Our system: Rhizoma

- CLP solver calculates better deployment



Our system: Rhizoma

- Actuator acquires/releases VMs, starts/stops application



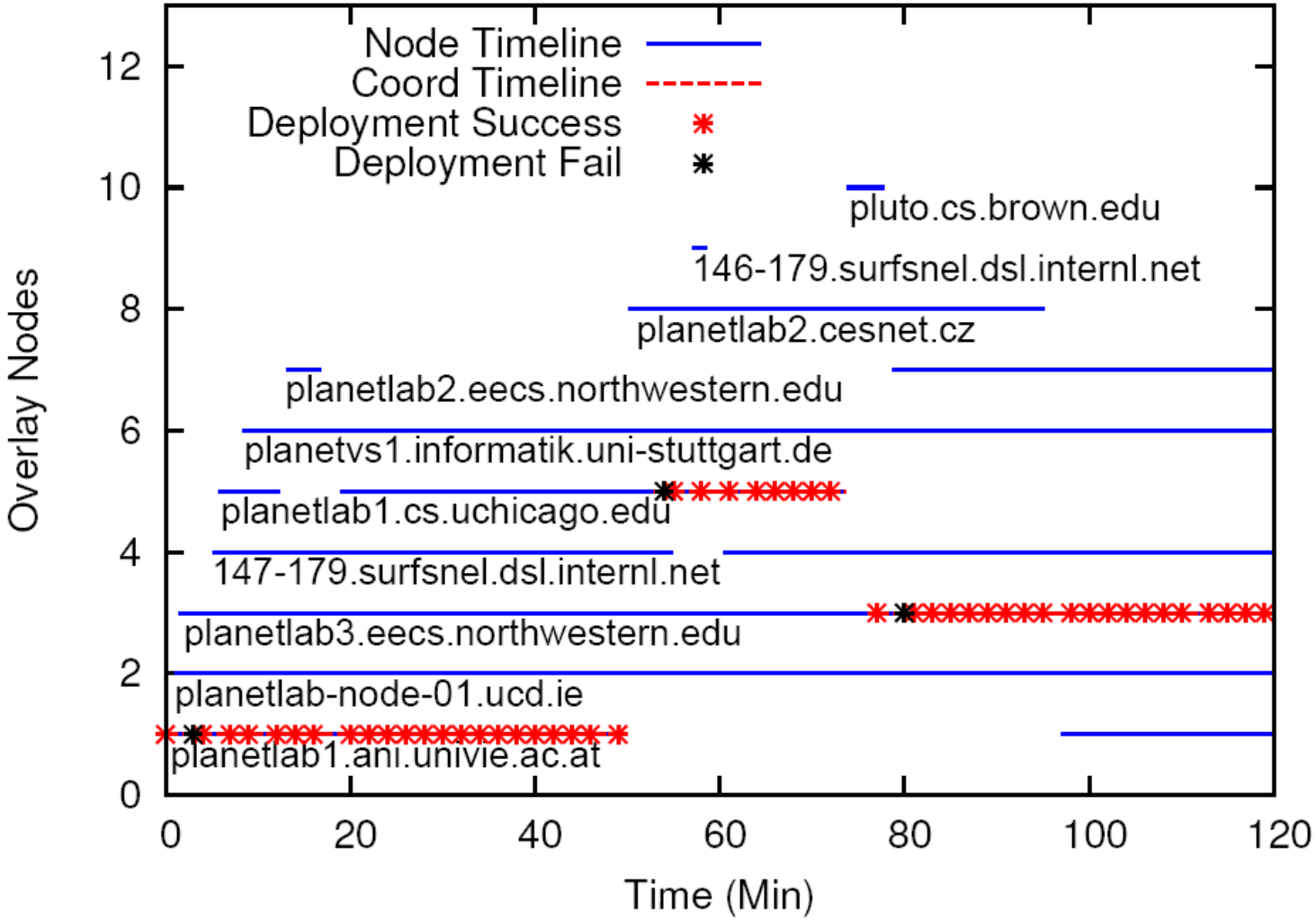
Evaluation

- Rhizoma runtime
- PsEPR requirements

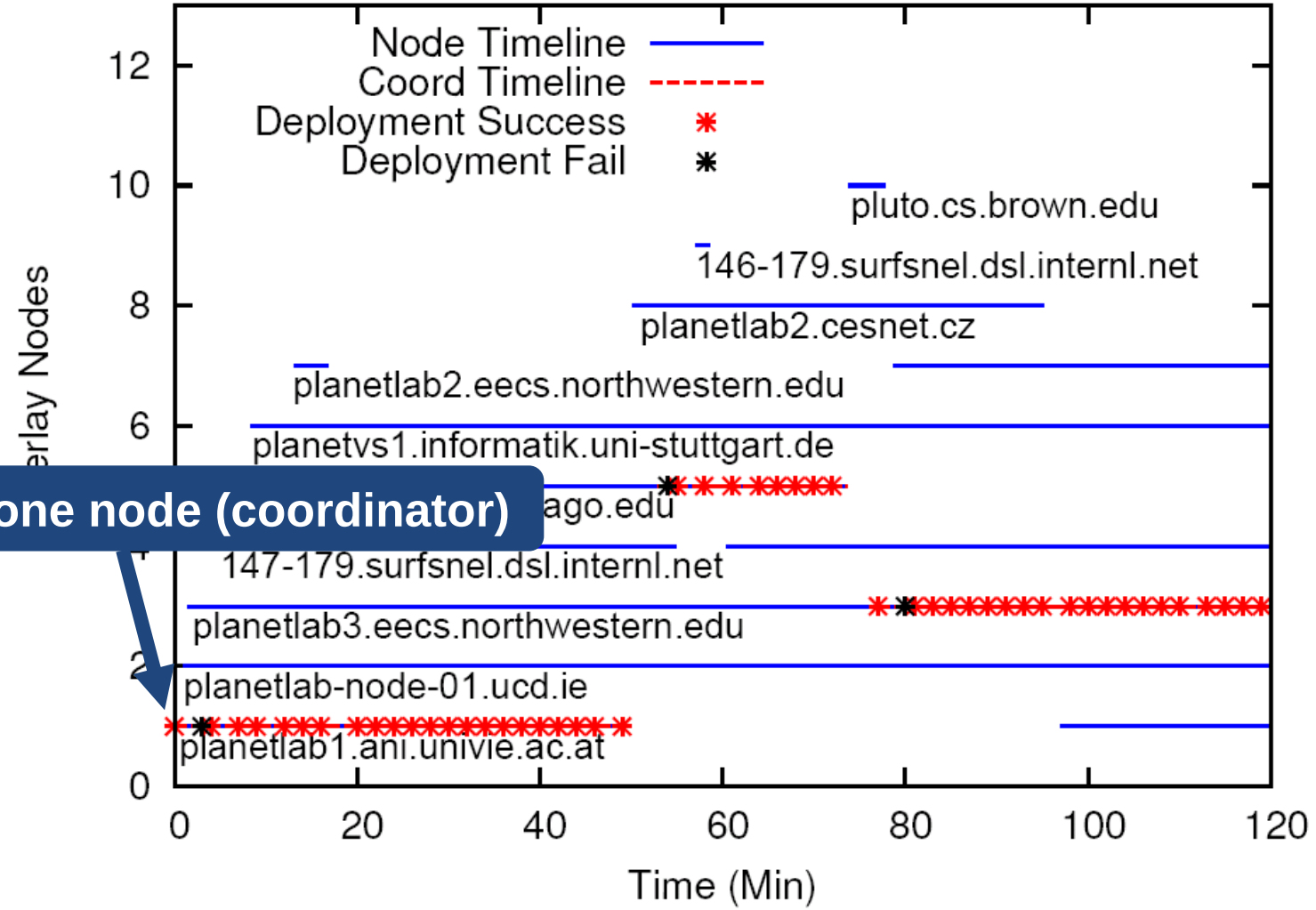
- 8 hour trace running on PlanetLab

- Overlay behavior
- Utility value
- Weighted attributes

Rhizoma hosting PsEPR on PlanetLab

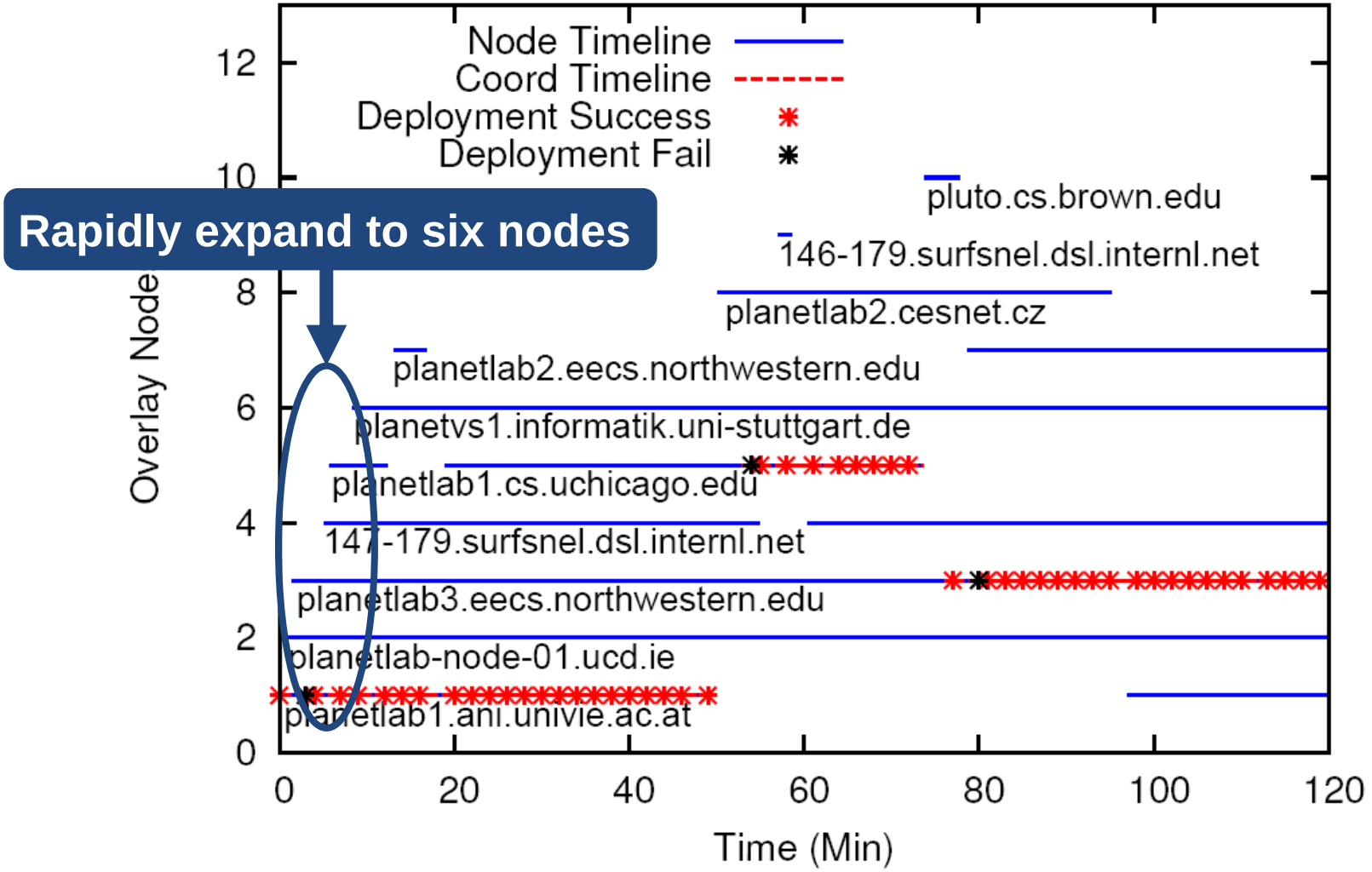


Rhizoma hosting PsEPR on PlanetLab

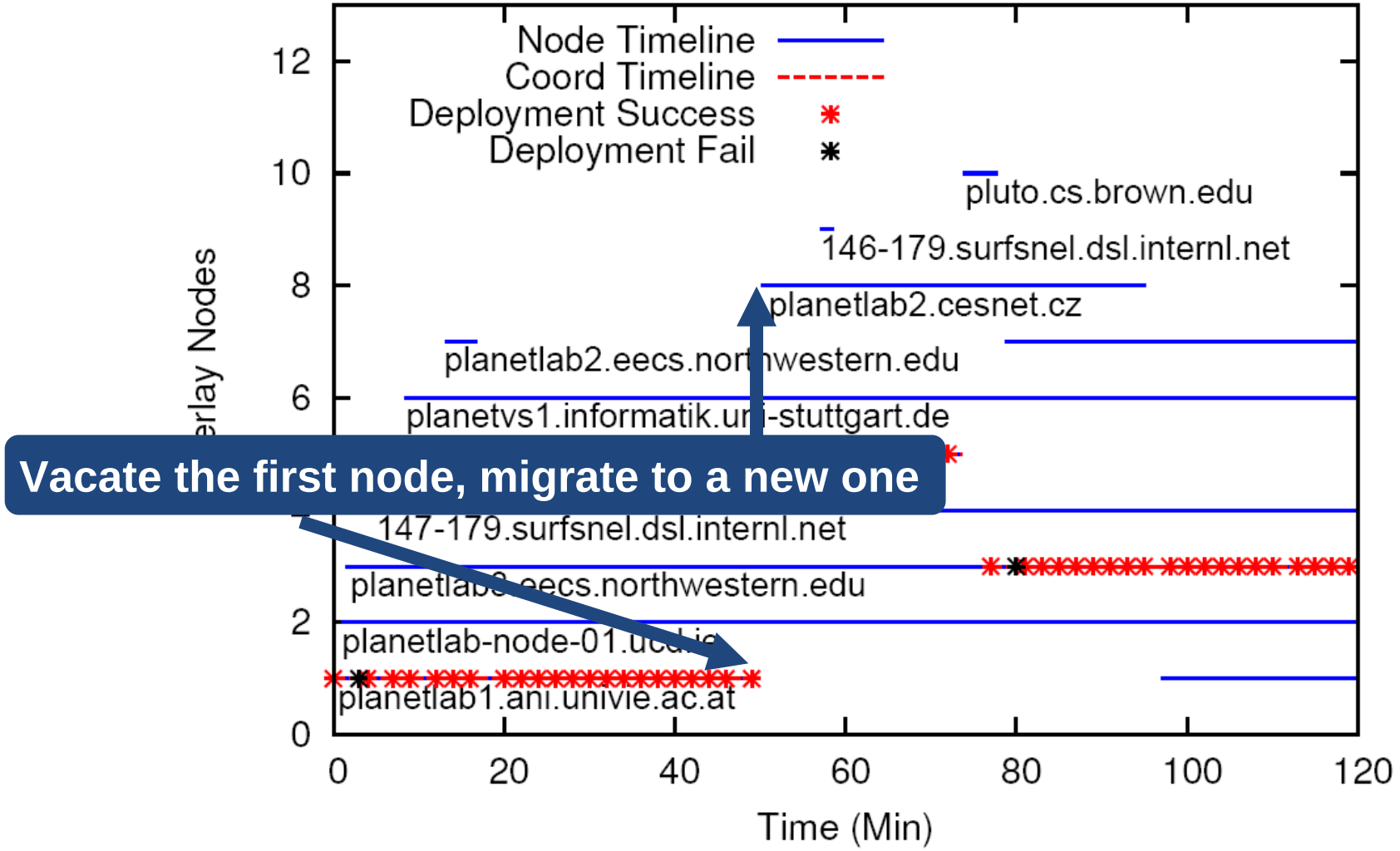


Start on one node (coordinator)

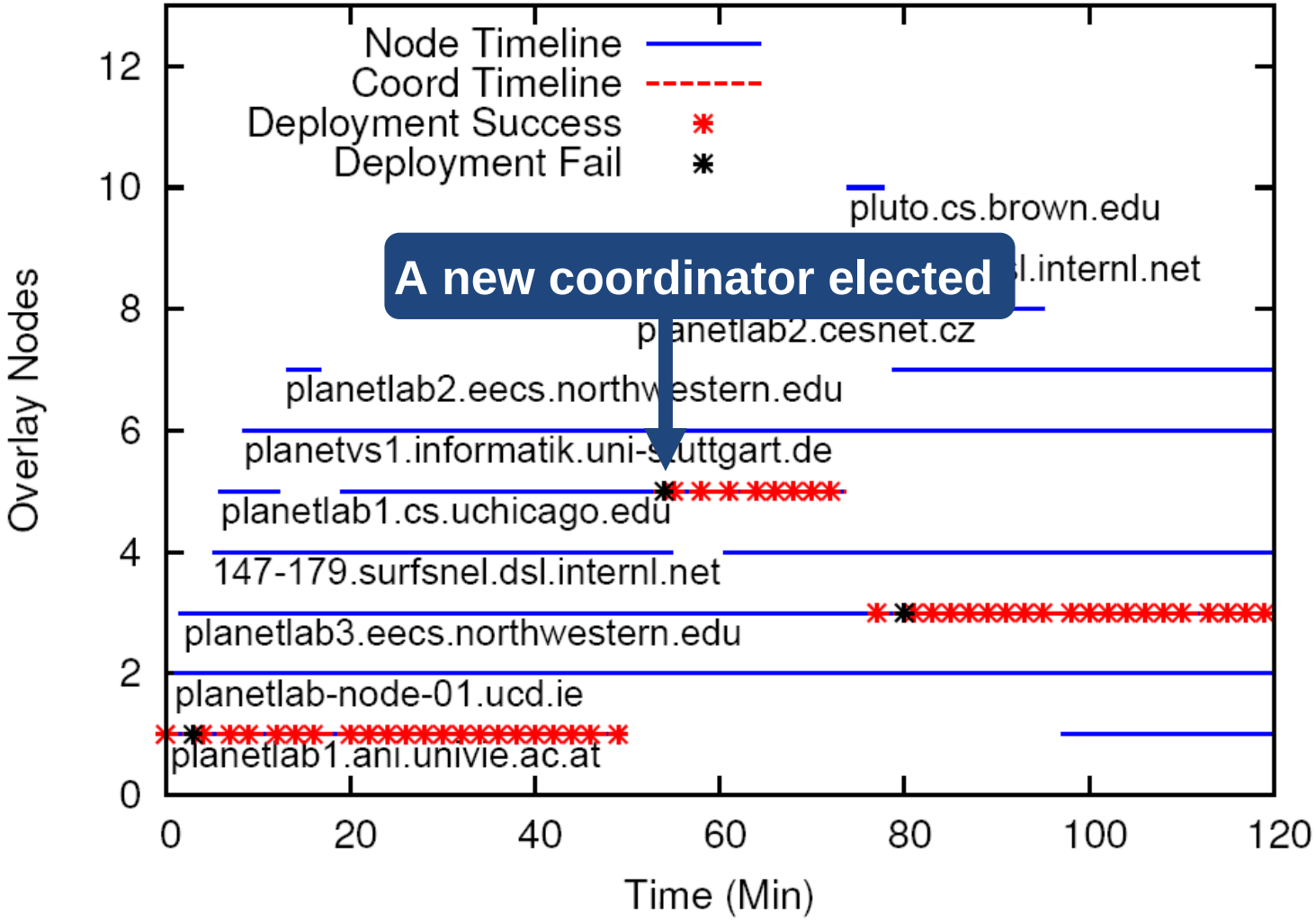
Rhizoma hosting PsEPR on PlanetLab



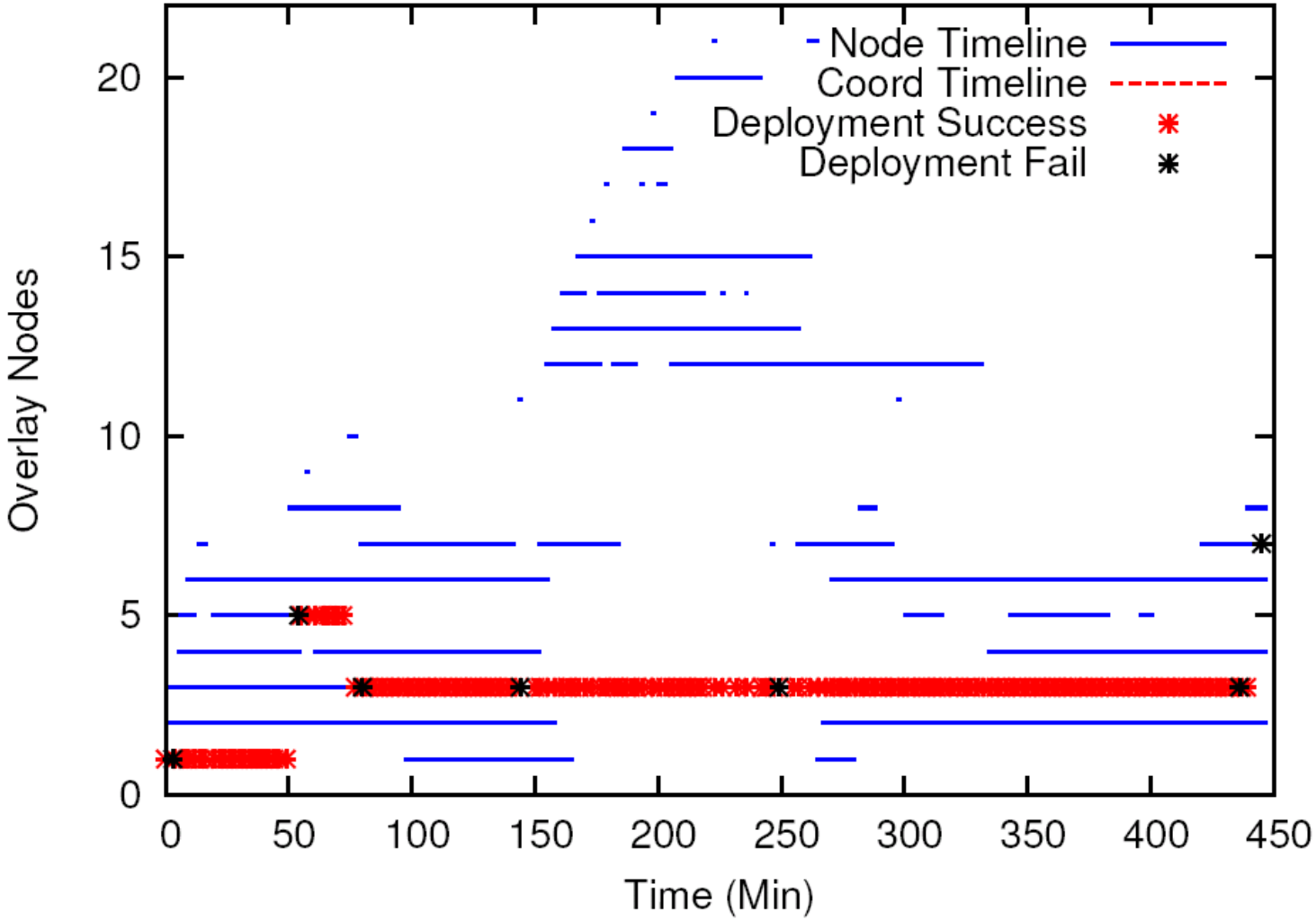
Rhizoma hosting PsEPR on PlanetLab



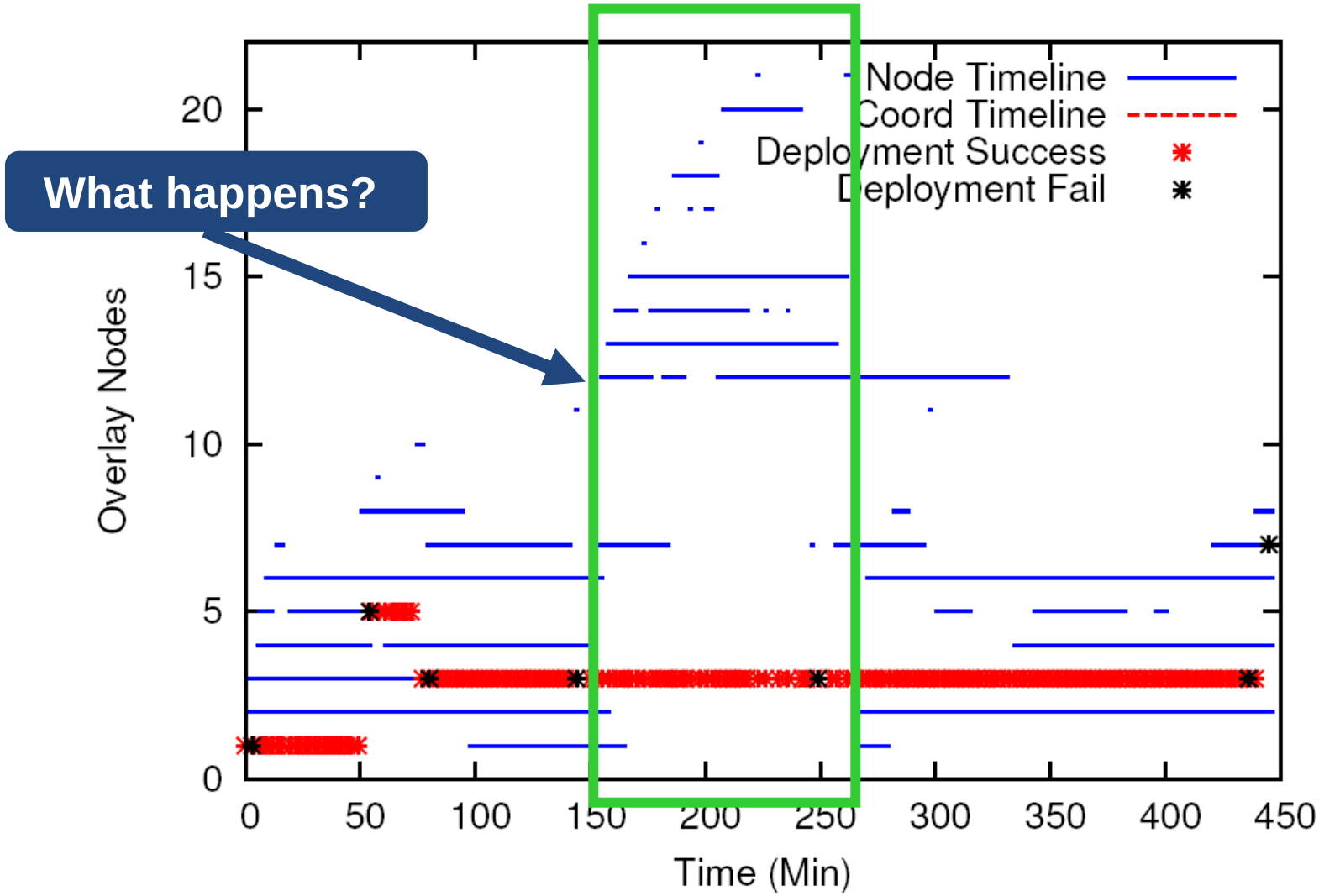
Rhizoma hosting PsEPR on PlanetLab



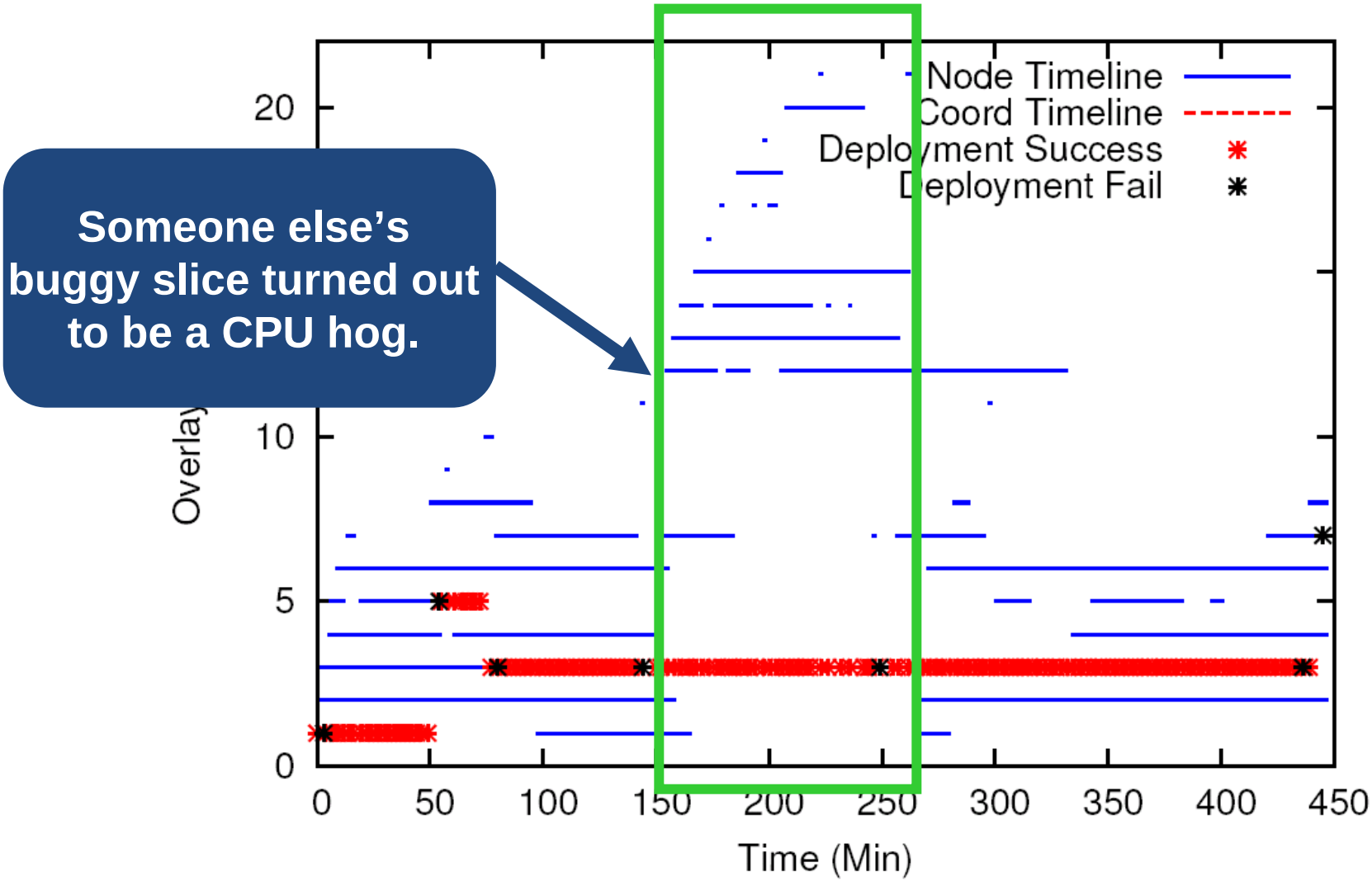
Rhizoma hosting PsEPR – longer trace



Rhizoma hosting PsEPR – longer trace

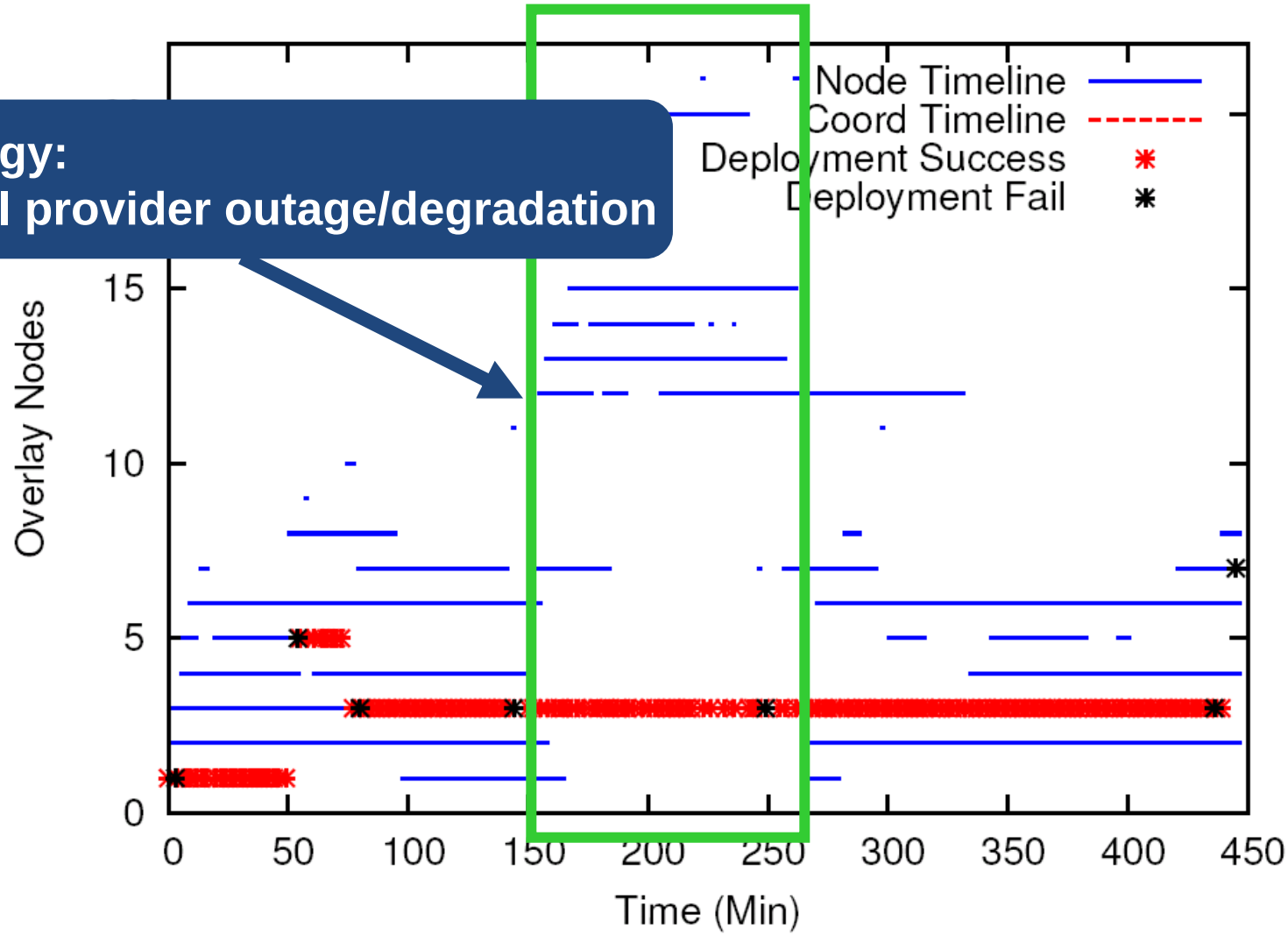


Rhizoma hosting PsEPR – longer trace

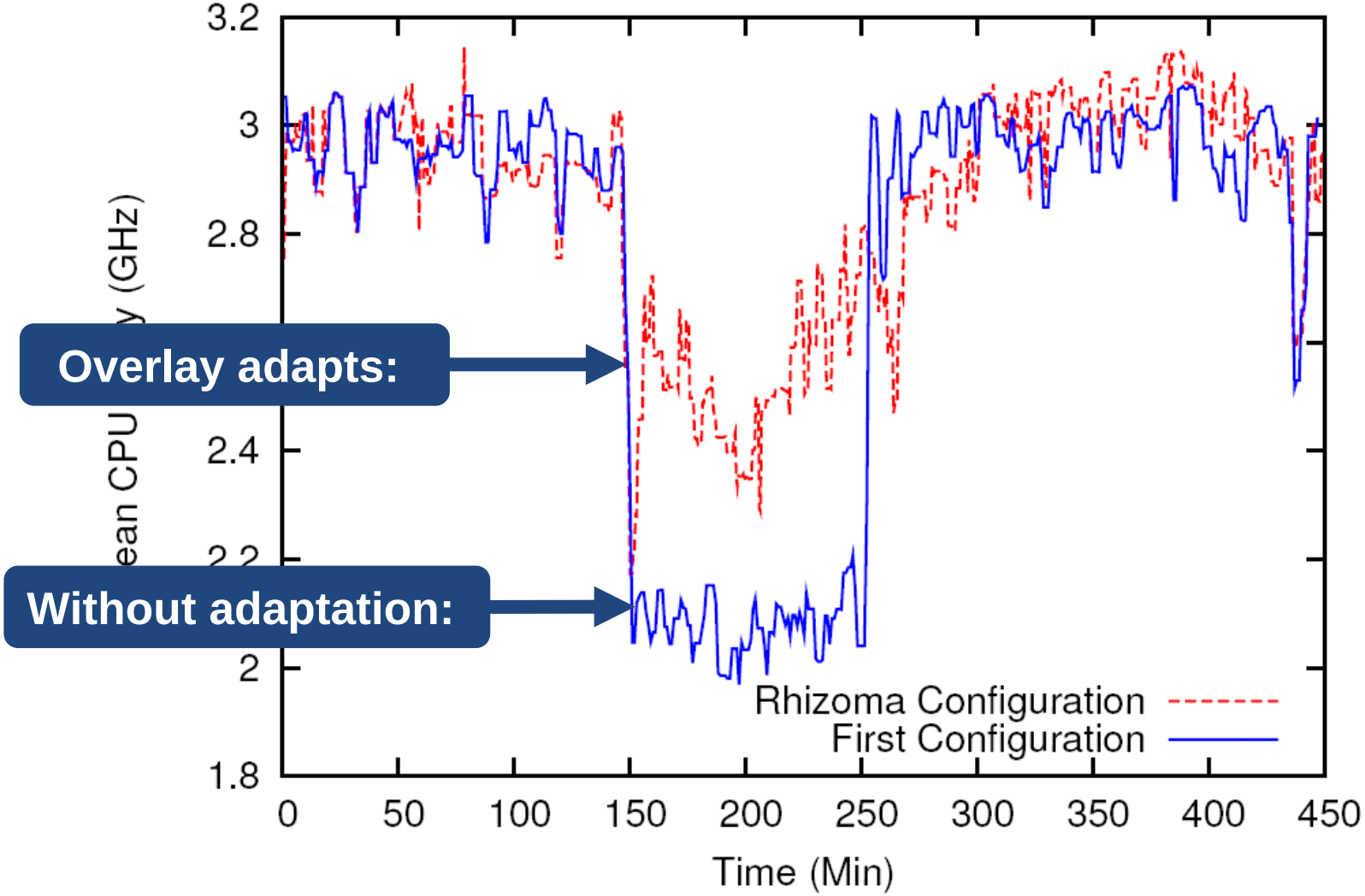


Rhizoma hosting PsEPR – longer trace

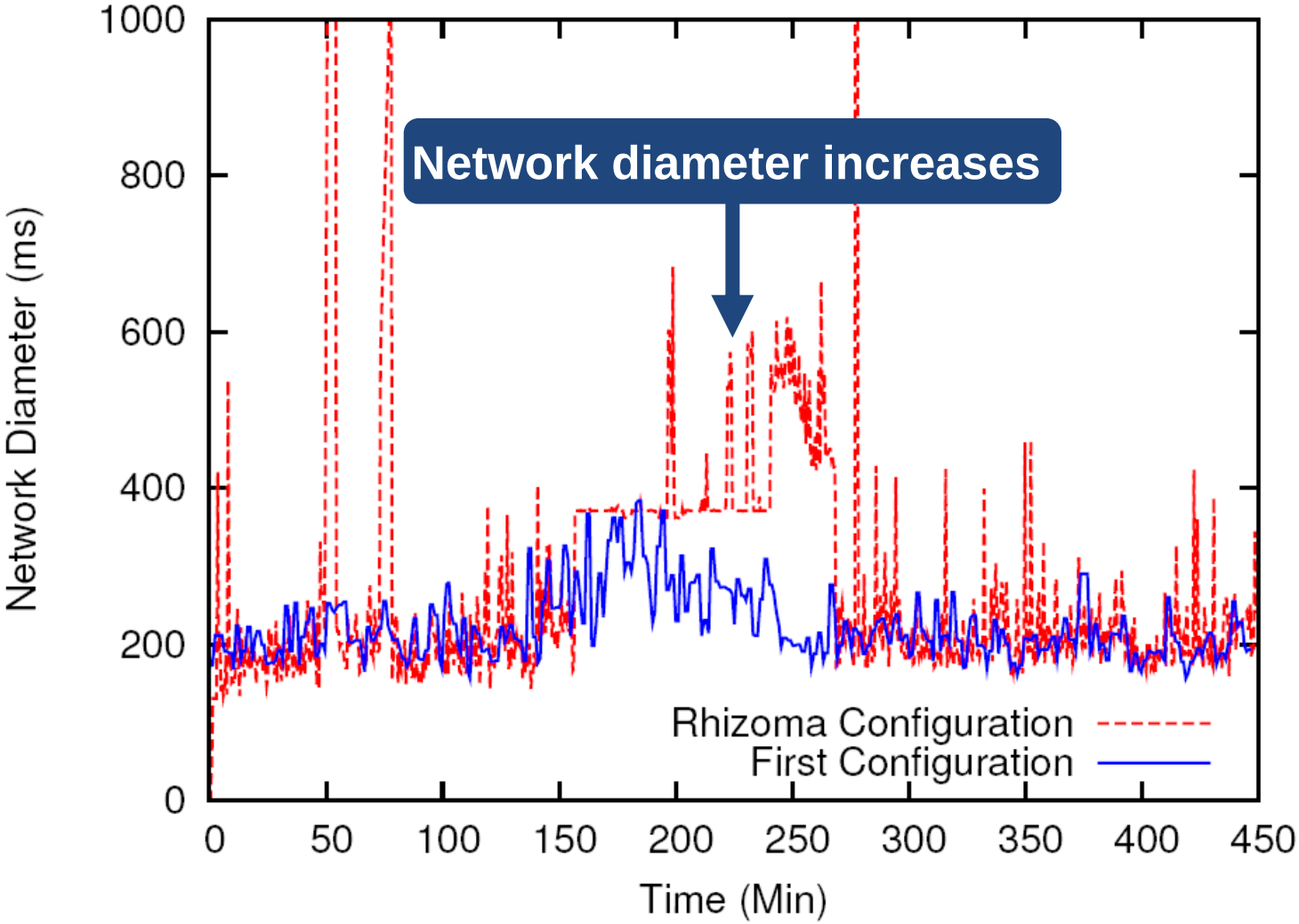
Analogy:
Partial provider outage/degradation



Mean CPU (GHz)

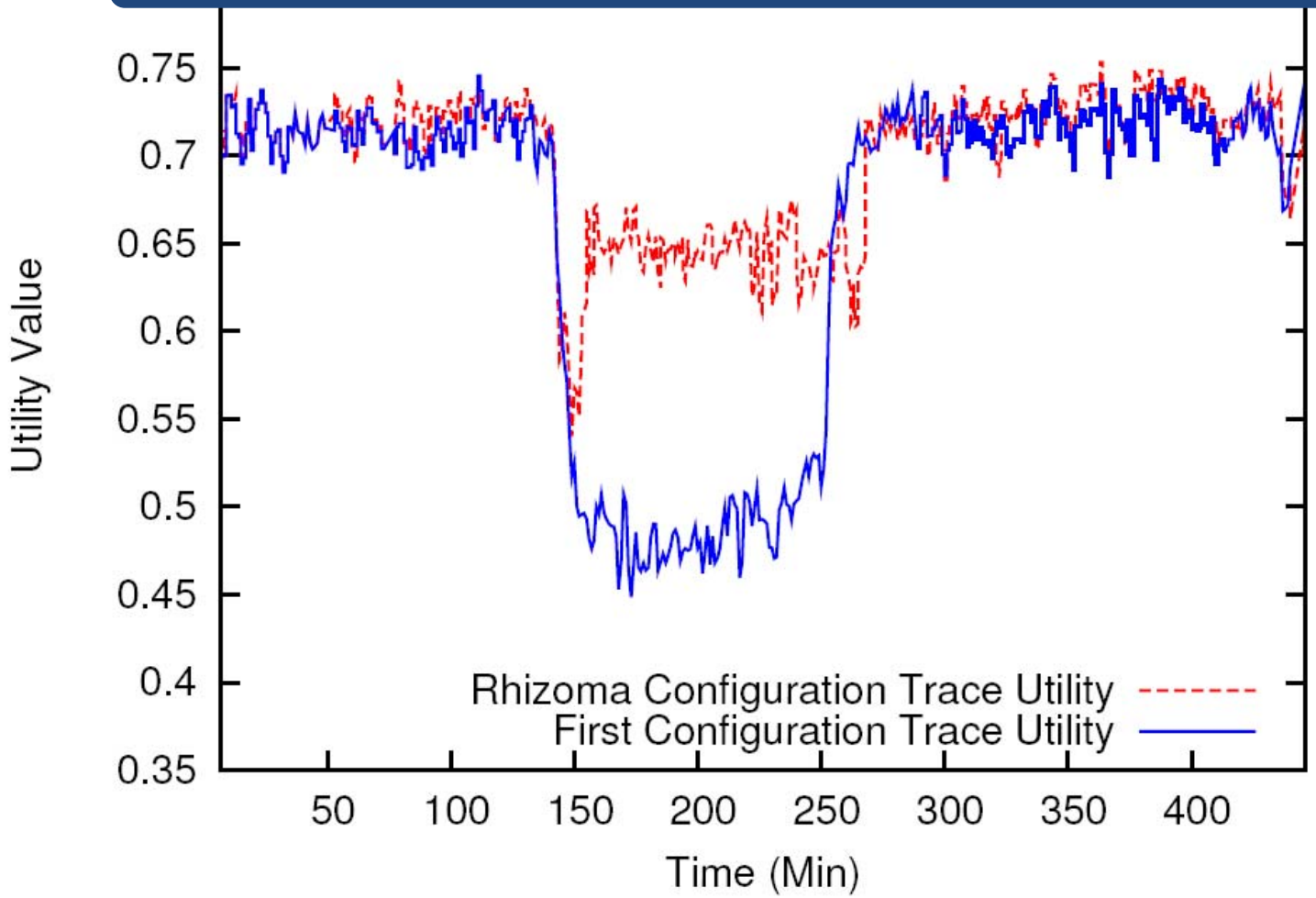

Overlay adapts:
Without adaptation:
3.2
3
2.8
2.4
2.2
2
1.8
0
50
100
150
200
250
300
350
400
450
Time (Min)
Rhizoma Configuration
First Configuration

Network diameter (ms)

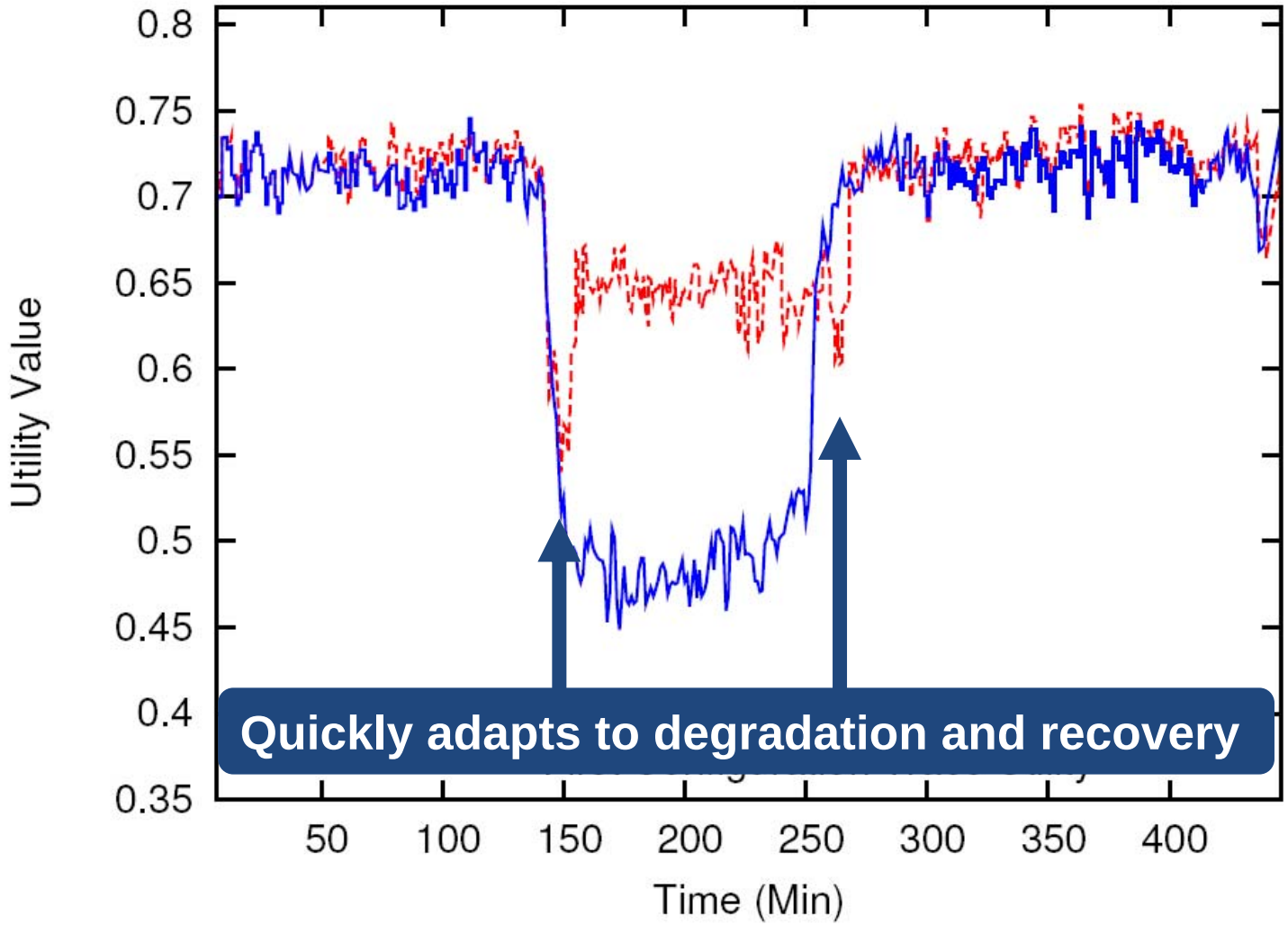


Utility values

Mean CPU is weighted more than network diameter



Utility values



Summary

- Key ideas:
  - **Runtime** for **self-deploying, self-managing** overlays
  - **CLP** to express desired application behavior

Summary

- Key ideas:
  - **Runtime** for **self-deploying, self-managing** overlays
  - **CLP** to express desired application behavior

- Appealing features:
  - **Expressiveness** of applications' resource requirements
  - **Optimization** in terms of performance and cost
  - **Responsiveness** to changes in resources, load, policies

Summary

- Key ideas:
  - **Runtime** for **self-deploying, self-managing** overlays
  - **CLP** to express desired application behavior
- Appealing features:
  - **Expressiveness** of applications' resource requirements
  - **Optimization** in terms of performance and cost
  - **Responsiveness** to changes in resources, load, policies
- Future work:
  - Platform integration (cluster, PlanetLab, Amazon EC2)
  - Feed application-level metrics to optimization process