Towards An Auditing Language for Preventing Cascading Failures

Ennan Zhai and Ruzica Piskac

Department of Computer Science

Yale

Background

- Cloud services ensure reliability by redundancy:
  - Storing data redundantly
  - Replicating service states across multiple nodes

Background

- Cloud services ensure reliability by redundancy:
  - Storing data redundantly
  - Replicating service states across multiple nodes
- Examples:
  - Microsoft Azure, Amazon AWS, Google, etc. replicate their data and service states

However, cloud outages still occur!

Correlated failures resulting from EBS due to bugs in one EBS server

Summary of the October 22, 2012 AWS Service Event in the US-East Region

We'd like to share more about the service event that occurred on Monday, October 22nd in the US-East Region. We have now completed the analysis of the events that affected AWS customers, and we want to describe what happened, our understanding of how customers were affected, and what we are doing to prevent a similar issue from occurring in the future.

The Primary Event and the Impact to Amazon Elastic Block Store (EBS) and Amazon Elastic Compute Cloud (EC2)

However, cloud outages still occur!

Correlated failures resulting from EBS due to bugs in one EBS server

Summary of t

We'd like to sha
in the US-East
AWS customers
customers were
in the future.

**The Primary
(EBS) and A**

Rackspace Outage Nov 12th

2 years ago 1,120 Views

On November 12th at 13:51 CST Rackspace experienced an isolated issue in their core network. A small number of their customers were affected, including REW. The outage lasted about 90 minutes. In simple terms, a core network switch died and when the traffic failed over to the secondary switch it also died. Rackspace is investigating the incident to find ways to improve their network and processes to ensure this event is not repeated. REW Sysadmins were immediately notified of the outage by our monitoring tools and were in constant contact with Rackspace during the outage working to resolve as quickly as possible.

REW apologizes for this outage; we promise that we are putting Rackspace's feet to the fire to ensure maximum uptime for our customers!

Here is the incident report from Rackspace if you want the techy details:

However, cloud outages still occur!

Final Root Cause Analysis and Improvement Areas: Nov 18 Azure Storage Service Interruption

Posted on December 17, 2014

Jason Zander, CVP, Microsoft Azure Team

On November 18, 2014, many of our Microsoft Azure customers experienced a service interruption that impacted Azure Storage and several other services, including Virtual Machines. Following the incident, we posted a blog that outlined a preliminary Root Cause Analysis (RCA), to ensure customers understood how we were working to address the issue. Since that time, our highest priority has been actively investigating and mitigating this incident. Today, we're sharing our final RCA, which includes a comprehensive outline of steps we've taken to mitigate against this situation happening again, as well as steps we're taking to improve our communications and support response. We sincerely apologize and recognize the significant impact this service interruption may have had on your applications and services. We appreciate the trust our customers place in Microsoft Azure, and I want to personally thank everyone for the feedback which will help our business continually improve.

Root Cause Analysis

On November 18th [PST] (November 19th [UTC]) Microsoft Azure experienced a service interruption that resulted in intermittent connectivity issues with the Azure Storage service in multiple regions. Dependent services, primarily

ng from EBS
erver

v 12th

issue in their core network. A
outage lasted about 90
traffic failed over to the
nt to find ways to improve their
ysadmins were immediately
contact with Rackspace during

kspace's feet to the fire to

Here is the incident report from Rackspace if you want the techy details:

However, cloud outages still occur!

Final Root Cause Analysis and
Im
Sto

Posted

On Nov
Azure S
outlined
address the issue. Since that time, our highest priority has been actively investigating and mitigating this incident. Today, we're sharing our final RCA, which includes a comprehensive outline of steps we've taken to mitigate against this situation happening again, as well as steps we're taking to improve our communications and support response. We sincerely apologize and recognize the significant impact this service interruption may have had on your applications and services. We appreciate the trust our customers place in Microsoft Azure, and I want to personally thank everyone for the feedback which will help our business continually improve.

Root Cause Analysis

On November 18th [PST] (November 19th [UTC]) Microsoft Azure experienced a service interruption that resulted in intermittent connectivity issues with the Azure Storage service in multiple regions. Dependent services, primarily

Why replications do not help?

ng from EBS

A
outage lasted about 90
traffic failed over to the
nt to find ways to improve their
ysadmins were immediately
contact with Rackspace during

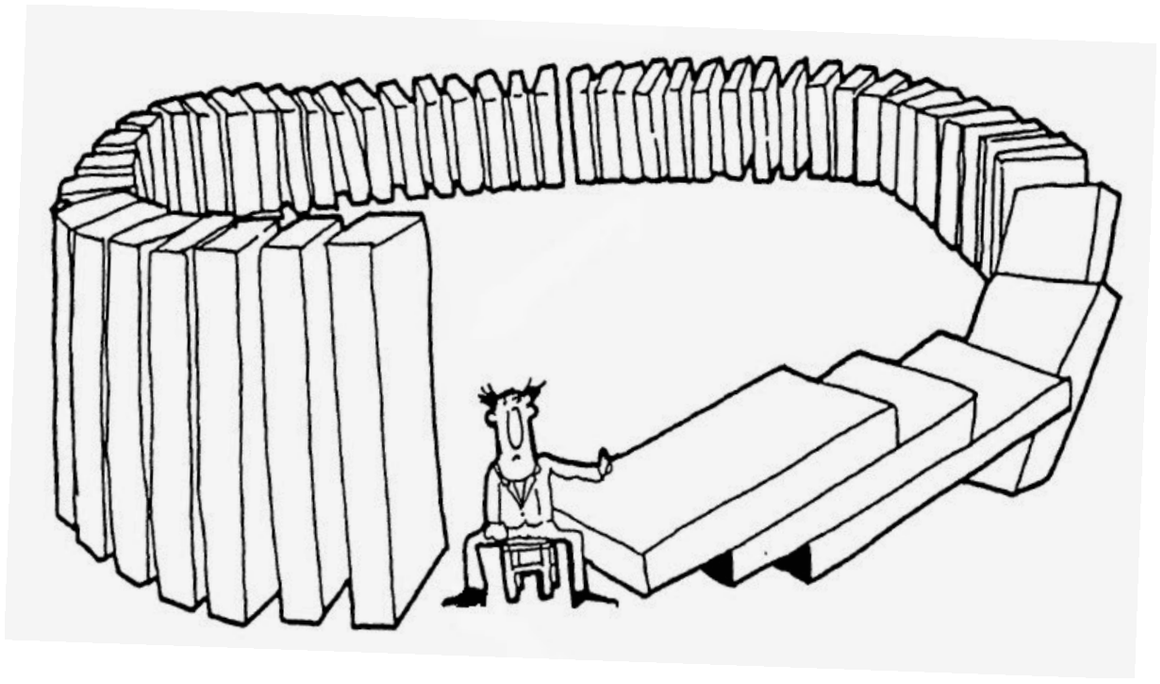
ckspace's feet to the fire to

Here is the incident report from Rackspace if you want the techy details:

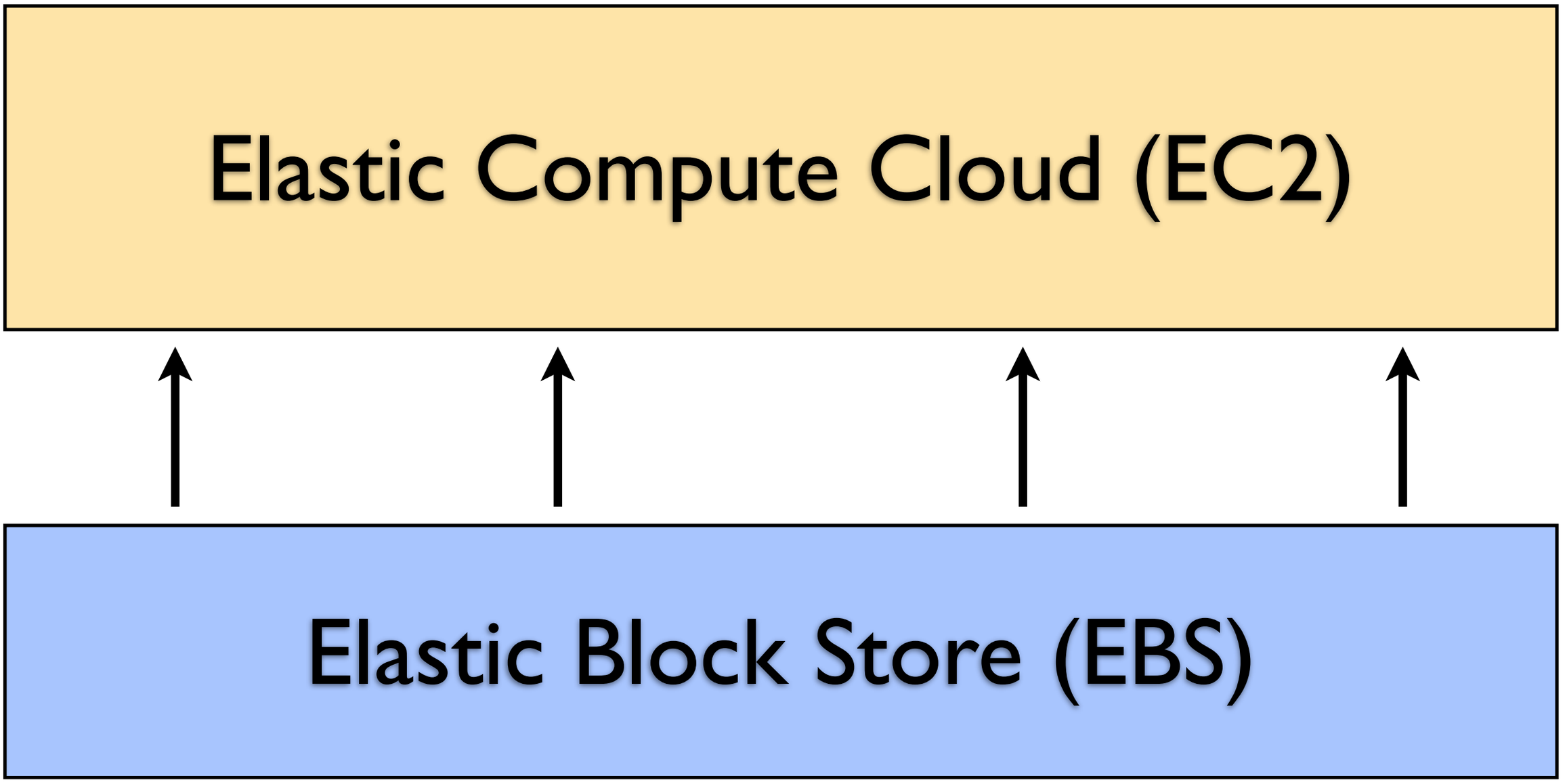
Cascading Failures!

[1] Why does the cloud stop computing? Lessons from hundreds of service outages

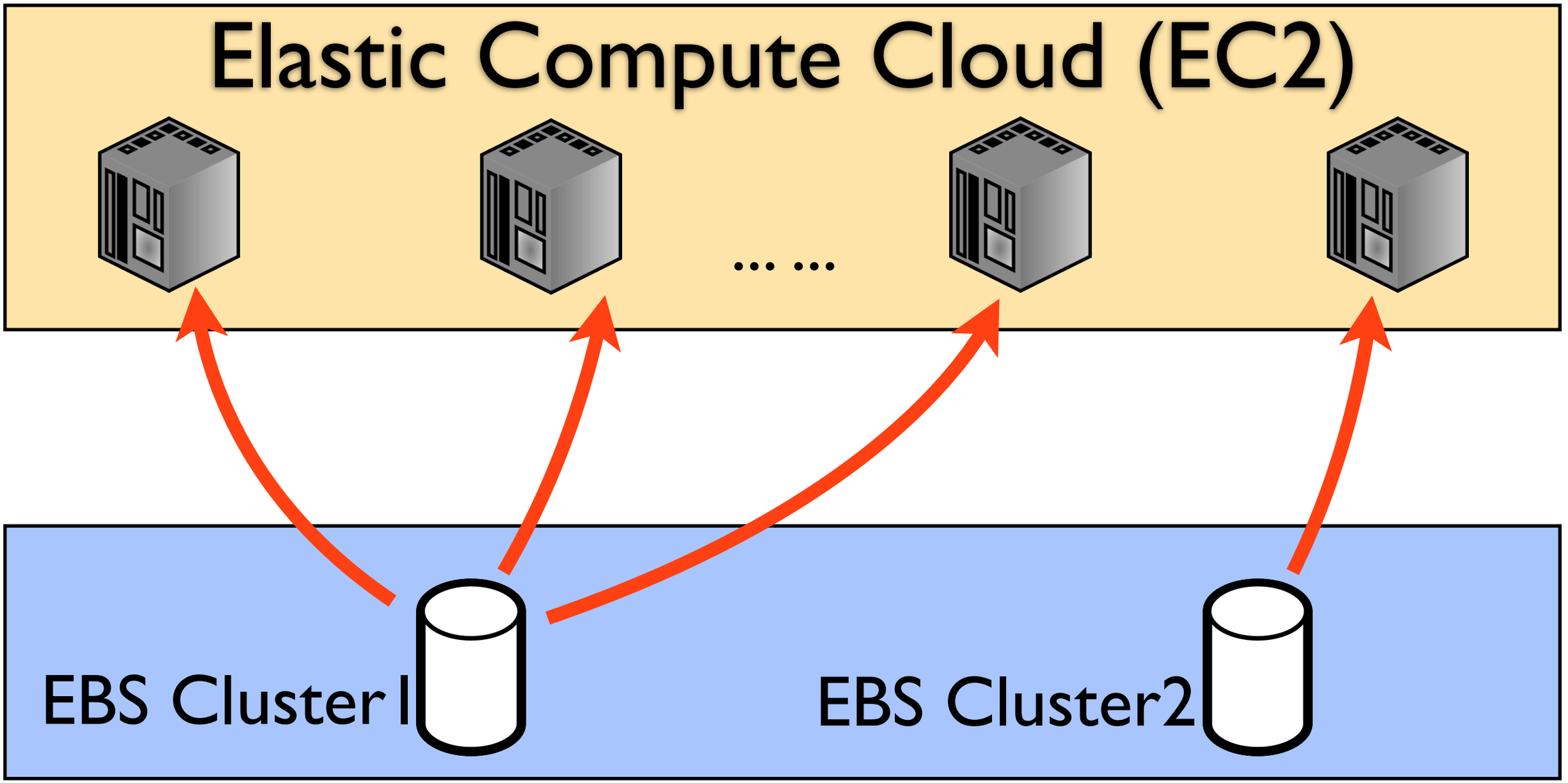
[2] What bugs live in the cloud? A study of 3000+ issues in cloud systems

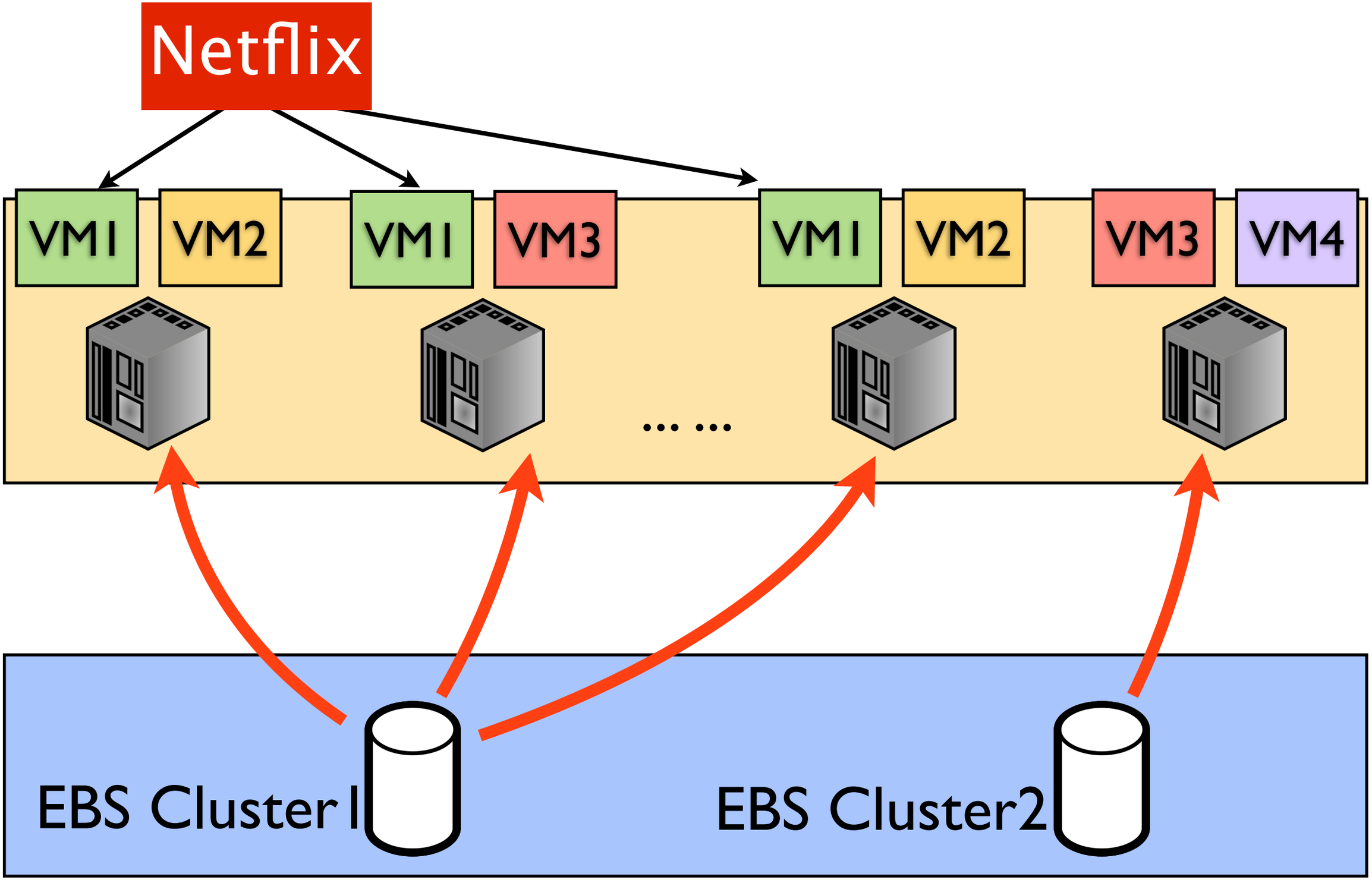


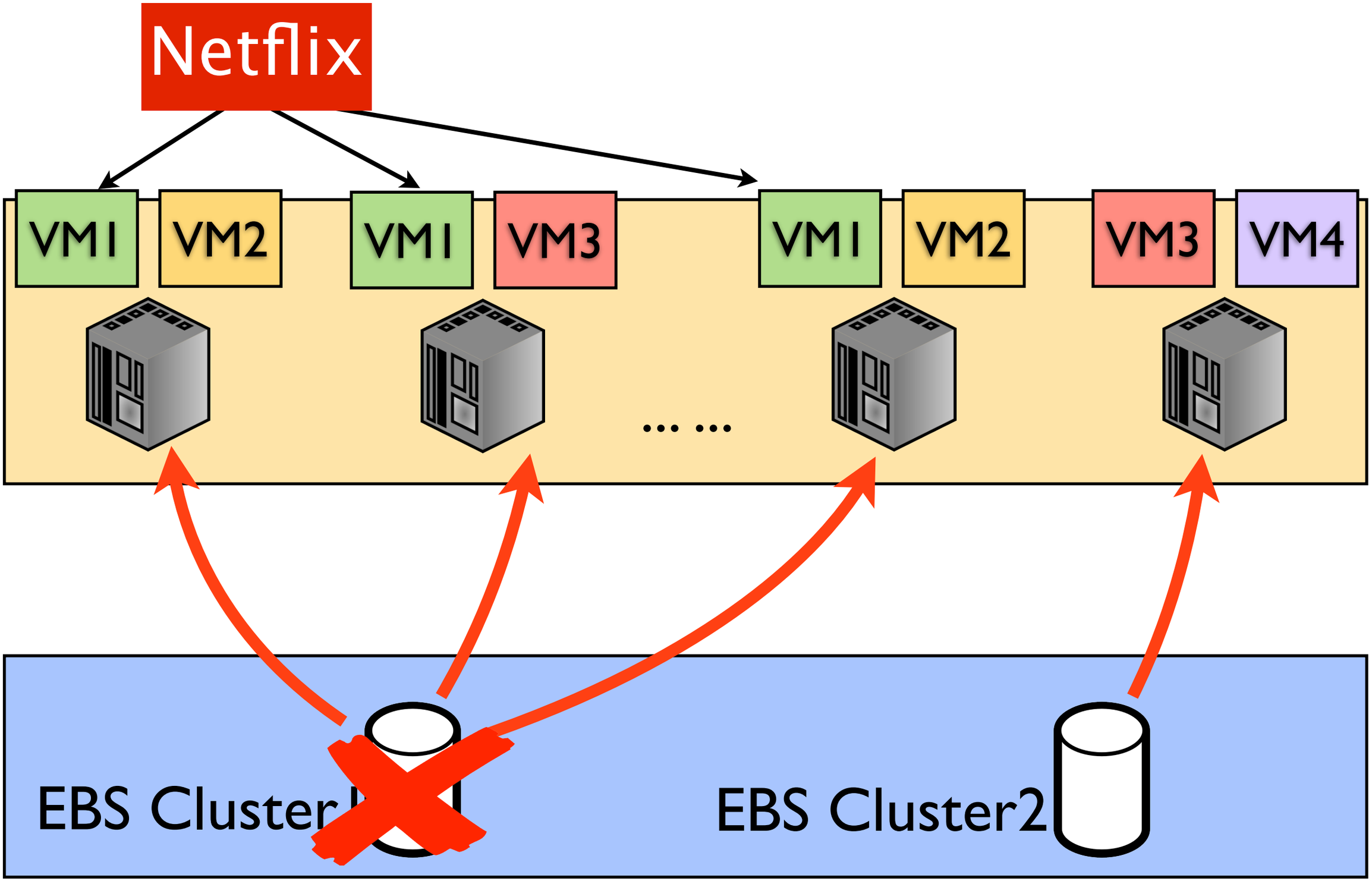
Cascading Failures in Reality

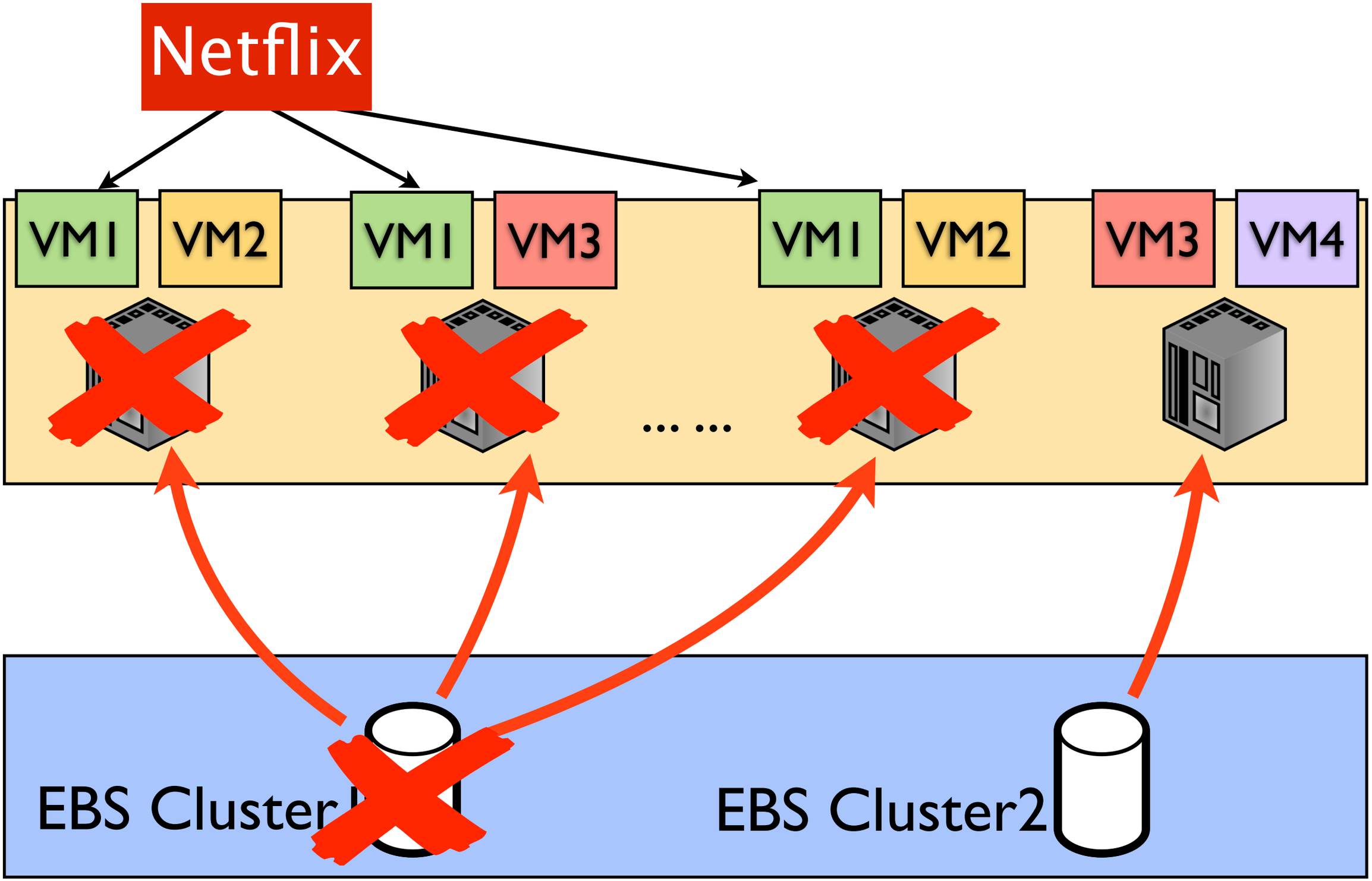


Cascading Failures in Reality

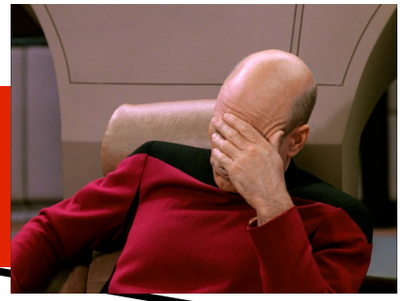



Netflix
VM1
VM2
VM1
VM3
VM1
VM2
VM3
VM4
EBS Cluster1
EBS Cluster2


Netflix
VM1
VM2
VM1
VM3
VM1
VM2
VM3
VM4
EBS Cluster1
EBS Cluster2


Netflix
VM1
VM2
VM1
VM3
VM1
VM2
VM3
VM4
EBS Cluster1
EBS Cluster2

Netflix



VM3

VM4

... ...

EBS Cluster1

EBS Cluster2

Existing Approaches

- Cloud providers handle cascading failures via:
  - Provenance systems (e.g., Y! [SIGCOMM'14] and ExSPAN [SIGMOD'10]);
  - Troubleshooting systems (e.g., Sherlock [SIGCOMM'07]).

Existing Approaches

- Cloud providers handle cascading failures via:
  - Provenance systems (e.g., Y! [SIGCOMM'14] and ExSPAN [SIGMOD'10]);
  - Troubleshooting systems (e.g., Sherlock [SIGCOMM'07]).
- Solving the problem *after* outage occurs.

Existing Approaches

- Cloud providers handle cascading failures via:
  - Provenance systems (e.g., Y! [SIGCOMM'14] and ExSPAN [SIGMOD'10]);
  - Troubleshooting systems (e.g., Sherlock [SIGCOMM'07]).
- Solving the problem *after* outage occurs.
- Prolonged recovery time in complex systems.

Existing Approaches

- Cloud providers handle cascading failures via:
  - Provenance systems (e.g., Y! [SIGCOMM'14] and ExSPAN [SIGMOD'10]);
  - Troubleshooting systems (e.g., Sherlock [SIGCOMM'07]).

- Solving the problem *after* outage occurs.
- Prolonged recovery time in complex systems.
- Cannot avoid system downtime.

Existing Approaches

- Cloud providers handle cascading failures via:
  - Provenance systems (e.g., X-[SIGCOMM'14] and
- Prolonged recovery time in complex systems.
- Cannot avoid system downtime.

Disease prevention is better than diagnosis

--World Health Organization

Our goal: Preventing cascading failures!

- INDaaS: First effort towards this goal [OSDI'14]

  Heading off correlated failures through Independence-as-a-Service

- An auditing language framework [OOPSLA'17]

  An auditing language for preventing correlated failures within the clouds


Service
initialization
Service Runtime

Post-Failure Forensics

1. Diagnosis tools
2. Accountability
3. Provenance
4. ... ...

Service initialization

Service Runtime

Our Initial Work

INDaaS
[OSDI'14]

Service
initialization

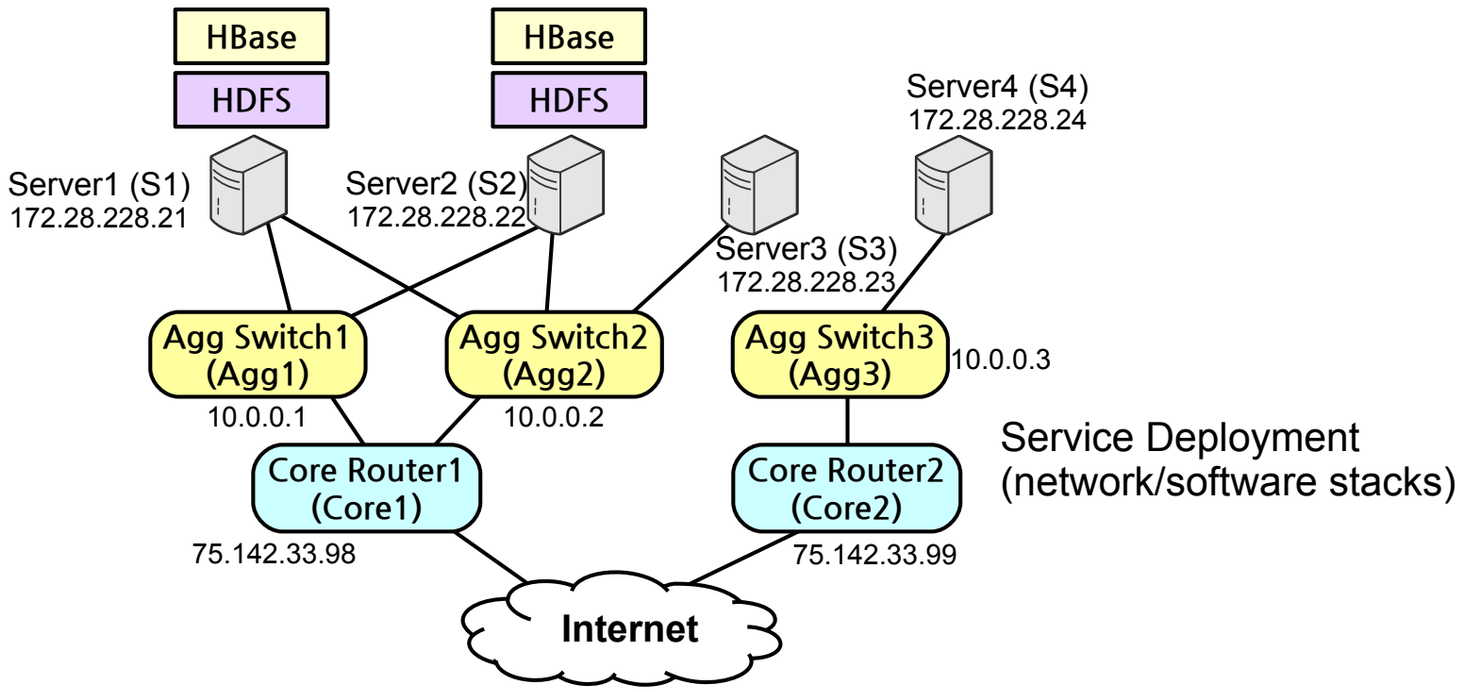
Post-Failure Forensics

1. Diagnosis tools
2. Accountability
3. Provenance
4. ... ...

Service Runtime

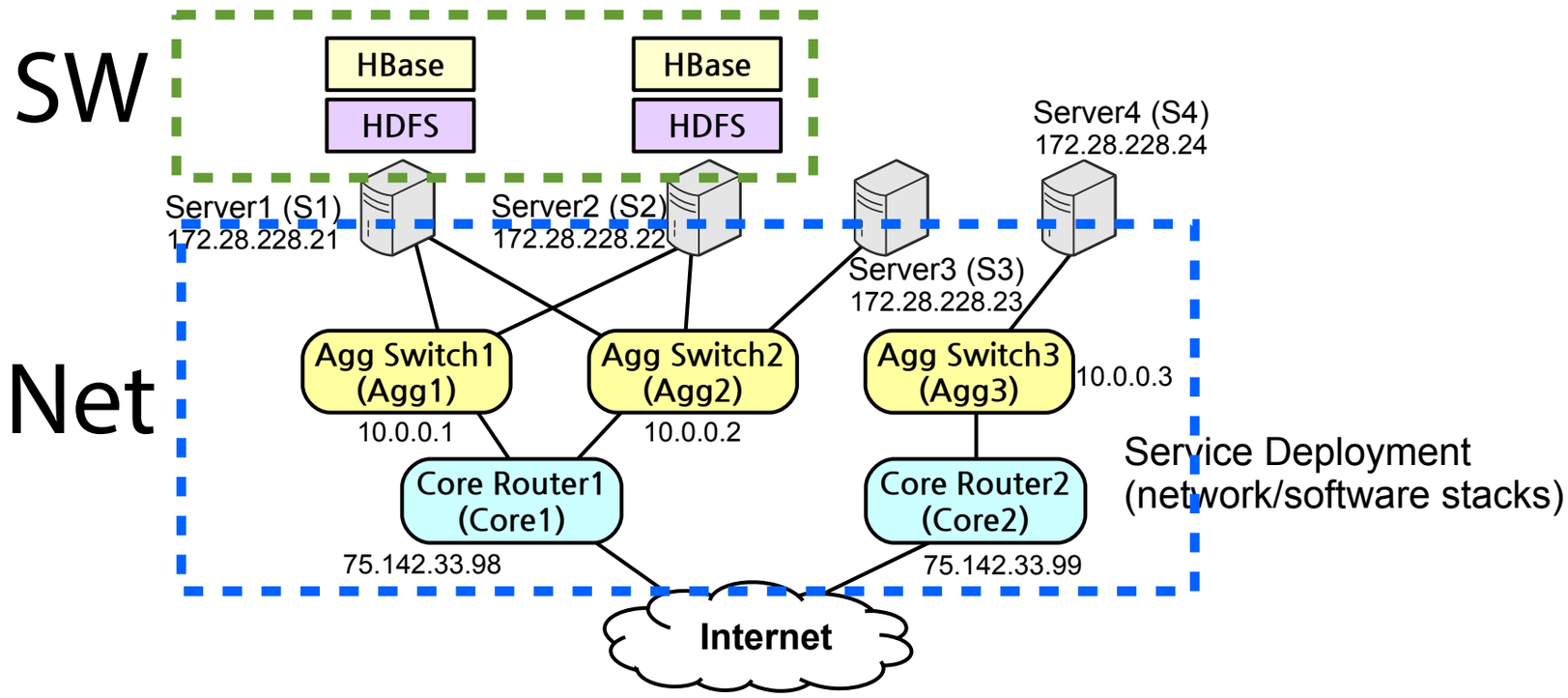
Our Initial Work: INDaaS [OSDI'14]

- INDaaS does pre-deployment recommendations:



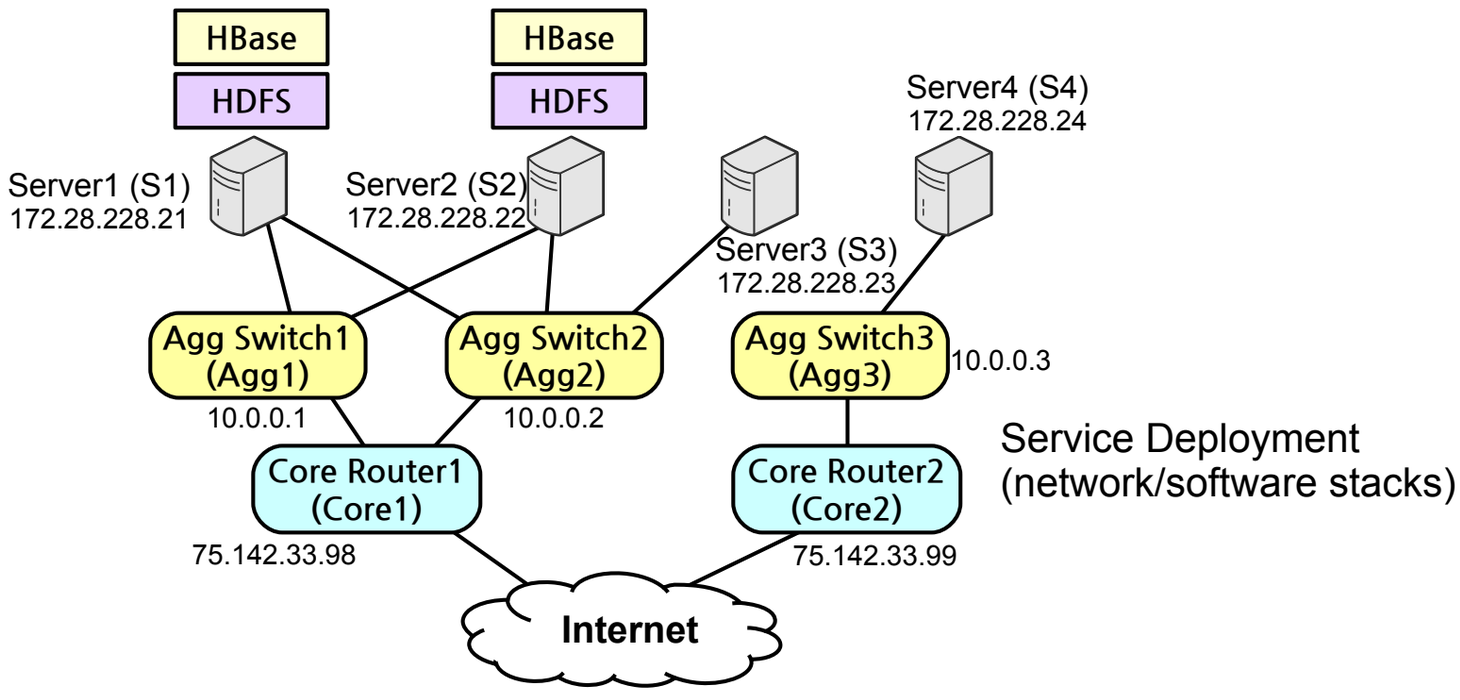
Our Initial Work: INDaaS [OSDI'14]

- INDaaS does pre-deployment recommendations:
  - Step1: Automatically collecting dependency data

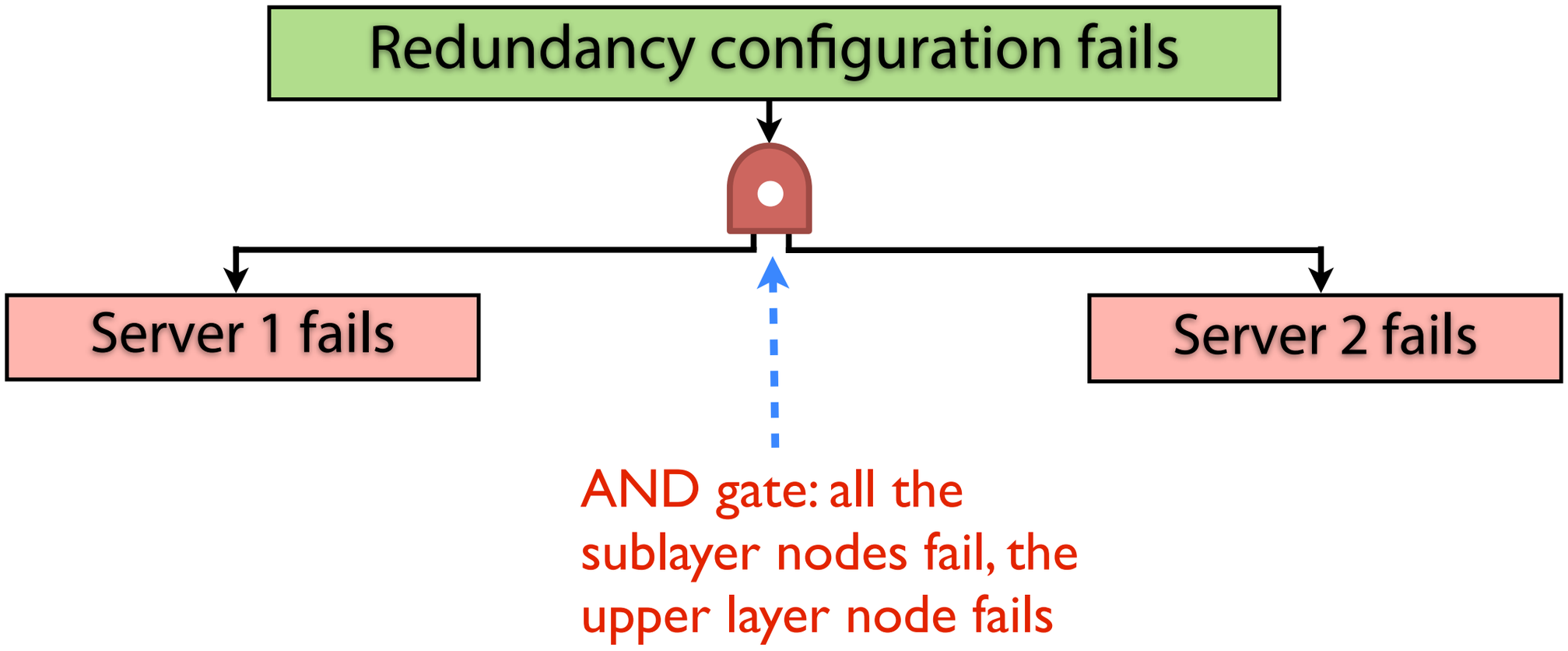


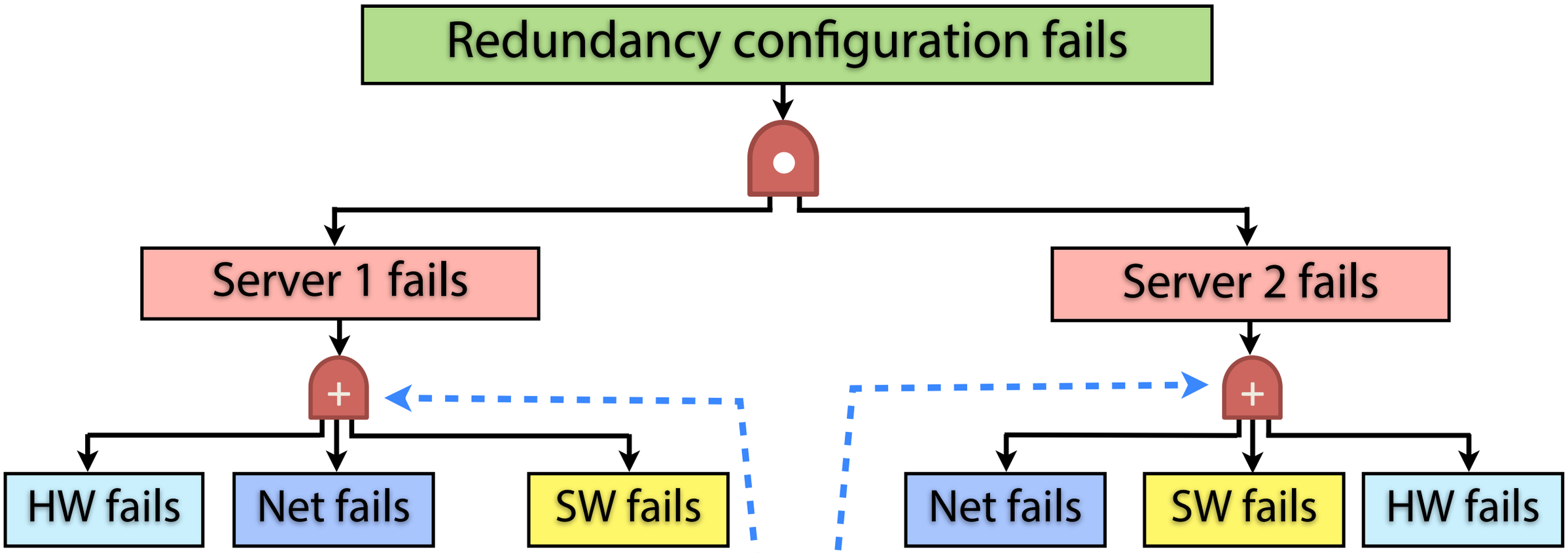
Our Initial Work: INDaaS [OSDI'14]

- INDaaS does pre-deployment recommendations:
  - Step1: Automatically collecting dependency data
  - Step2: Modeling system stack in fault graph
  - Step3: Evaluating independence of alternative redundancy configurations

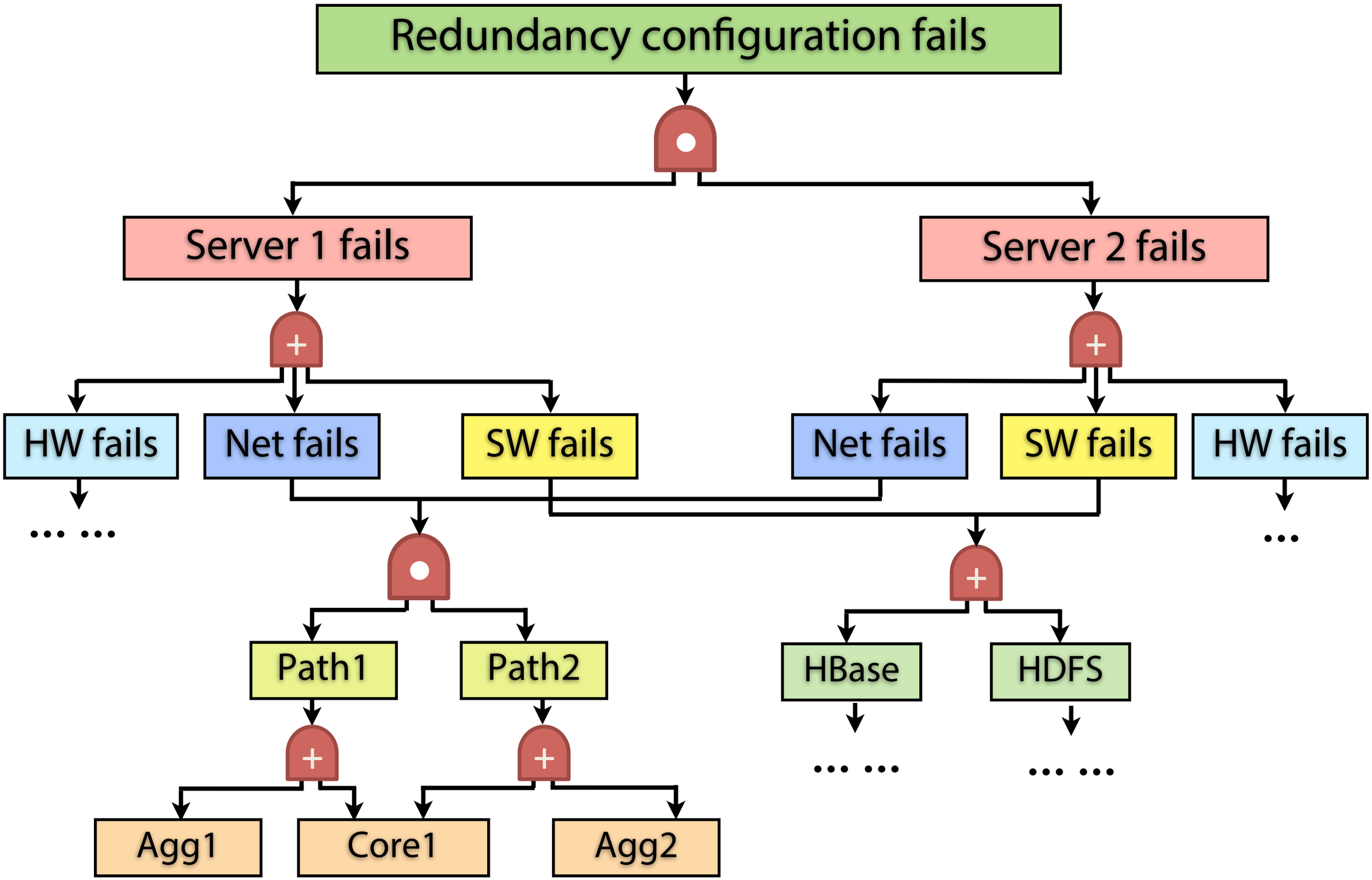


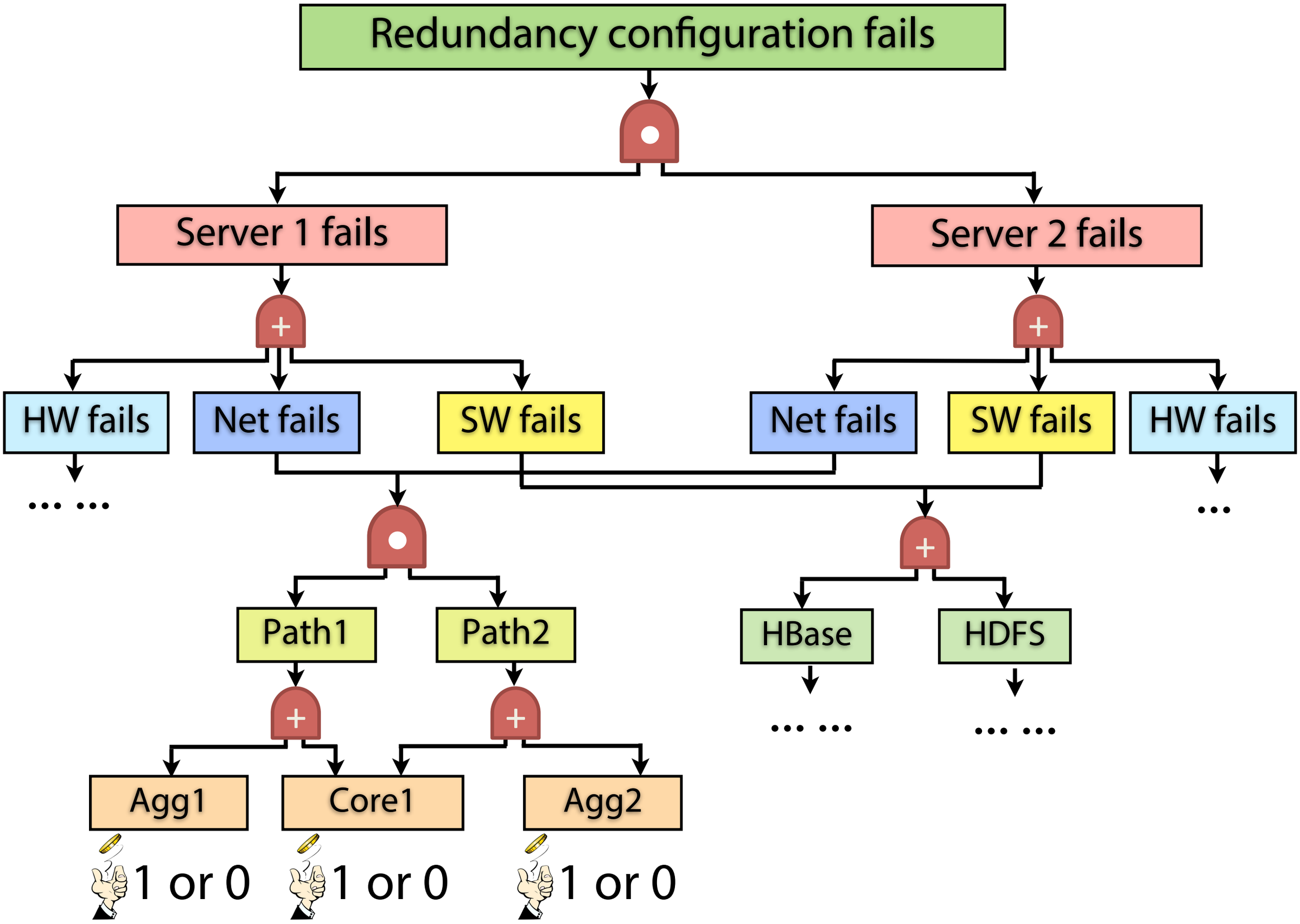
Redundancy configuration fails

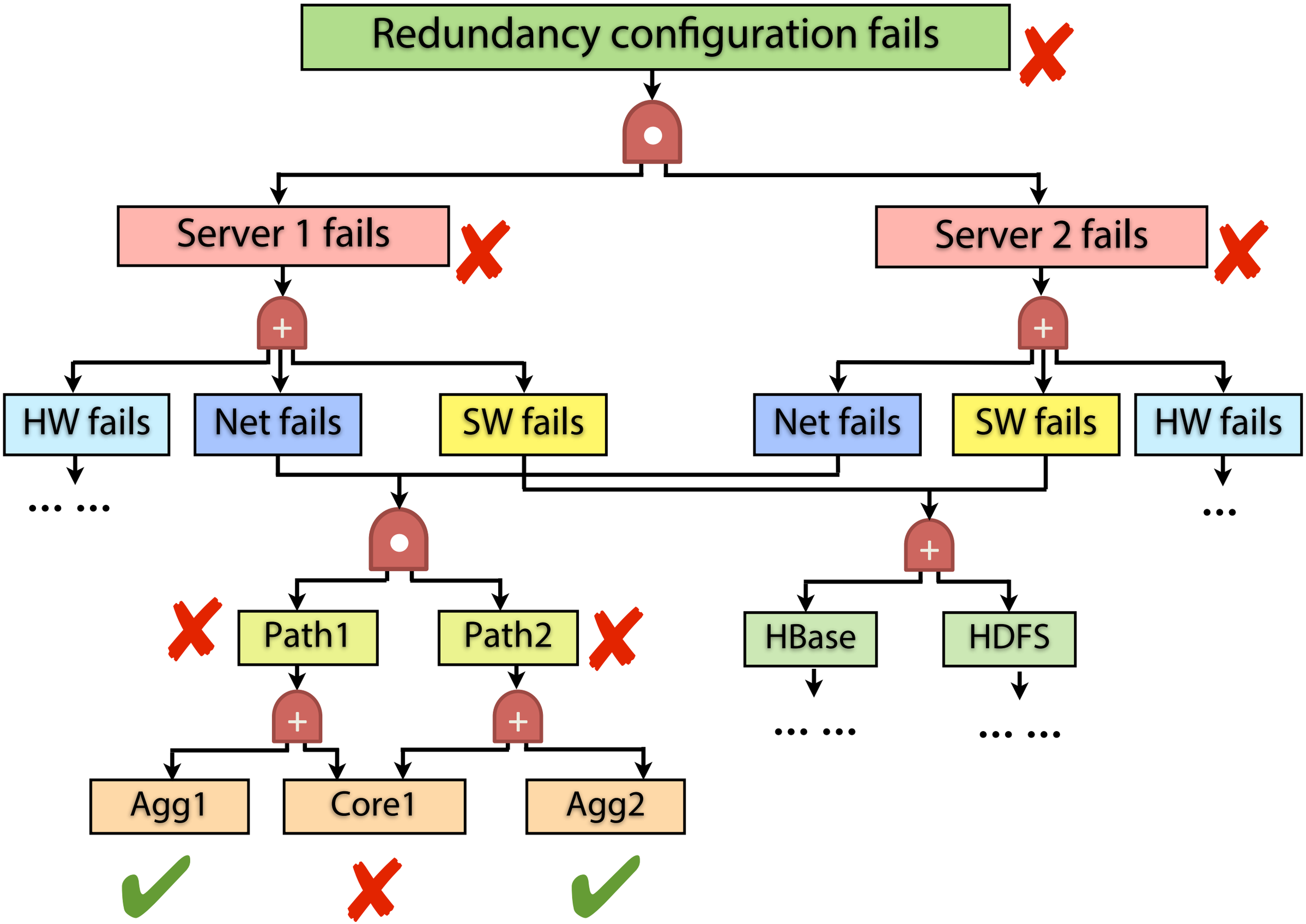

Redundancy configuration fails
Server 1 fails
Server 2 fails
AND gate: all the sublayer nodes fail, the upper layer node fails



OR gate: one of the sublayer nodes fails, the upper layer node fails


Redundancy configuration fails
Server 1 fails
Server 2 fails
HW fails
Net fails
SW fails
Net fails
SW fails
HW fails
Path1
Path2
HBase
HDFS
Agg1
Core1
Agg2


Redundancy configuration fails
Server 1 fails
Server 2 fails
HW fails
Net fails
SW fails
Net fails
SW fails
HW fails
Path1
Path2
HBase
HDFS
Agg1
Core1
Agg2
1 or 0
1 or 0
1 or 0


Redundancy configuration fails
Server 1 fails
Server 2 fails
HW fails
Net fails
SW fails
Net fails
SW fails
HW fails
Path1
Path2
HBase
HDFS
Agg1
Core1
Agg2

Issues in INDaaS

- Hard to express diverse auditing tasks, e.g., identifying risks

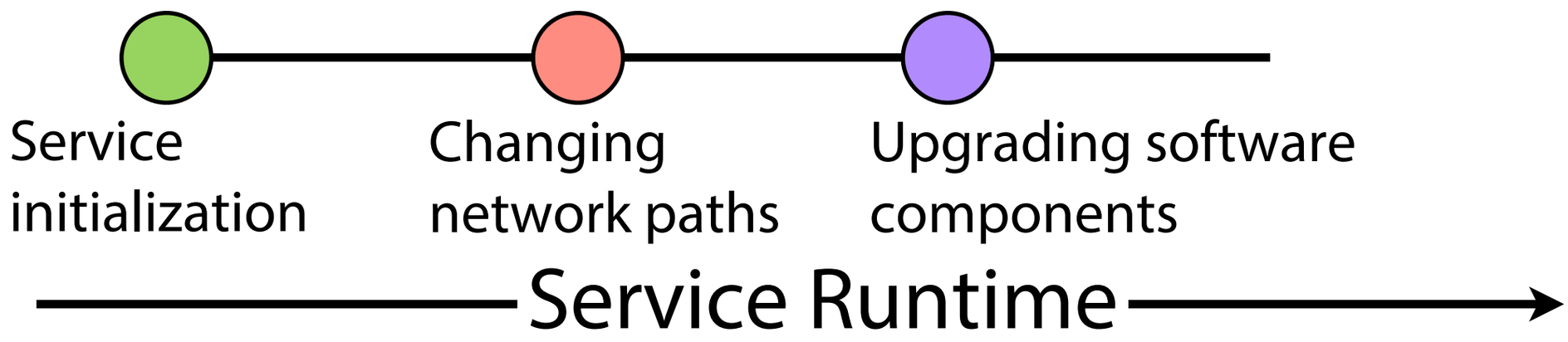
Issues in INDaaS

- Hard to express diverse auditing tasks, e.g., identifying risks
- Fault graph analysis does not support auditing in runtime

Issues in INDaaS

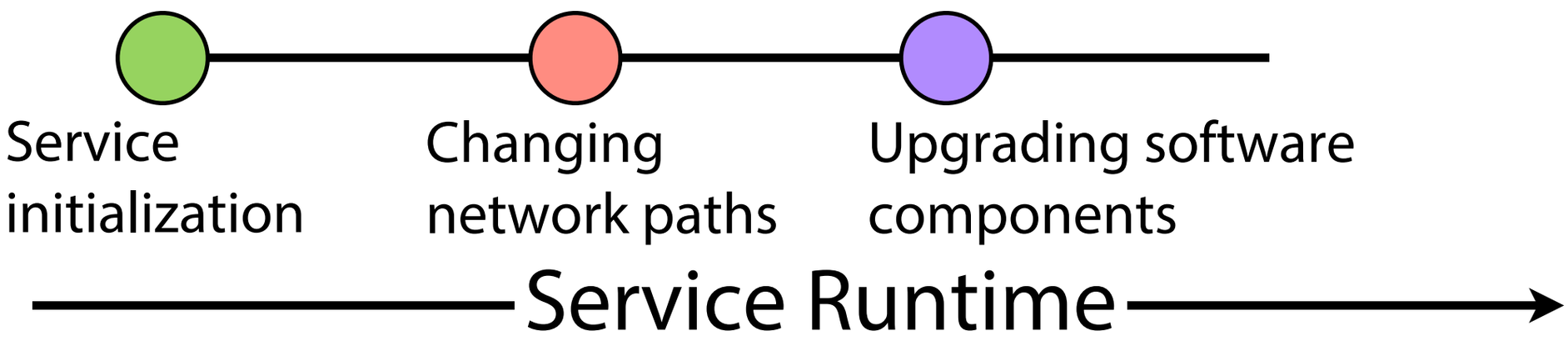
- Hard to express diverse auditing tasks, e.g., identifying risks

- Fault graph analysis does not support auditing in runtime



Issues in INDaaS

- Hard to express diverse auditing tasks, e.g., identifying risks
- Fault graph analysis does not support auditing in runtime
- Have no idea how to fix the cascading failure problem

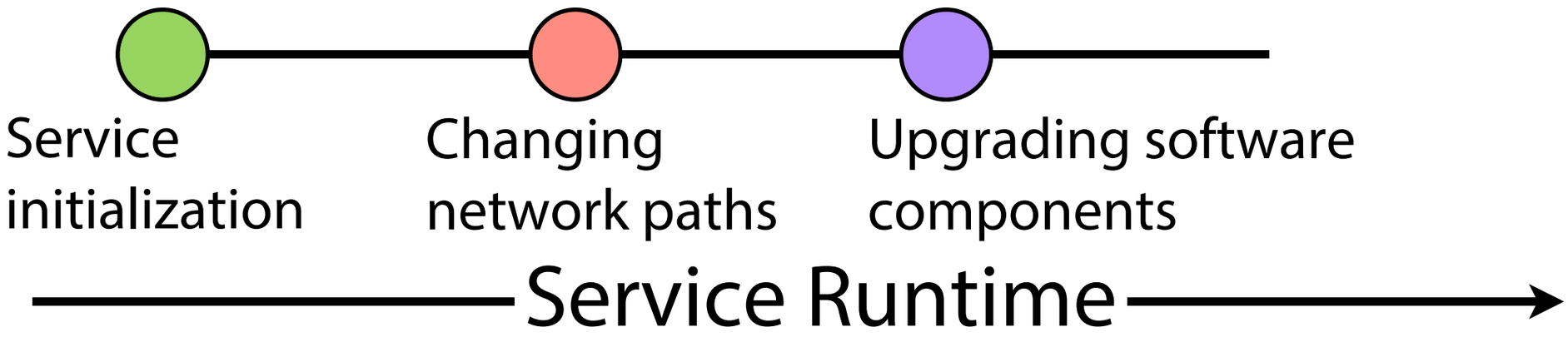


Issues in INDaaS

- Hard to express diverse auditing tasks, e.g., identifying risks
- Fa e
- Ha

Our Solution:

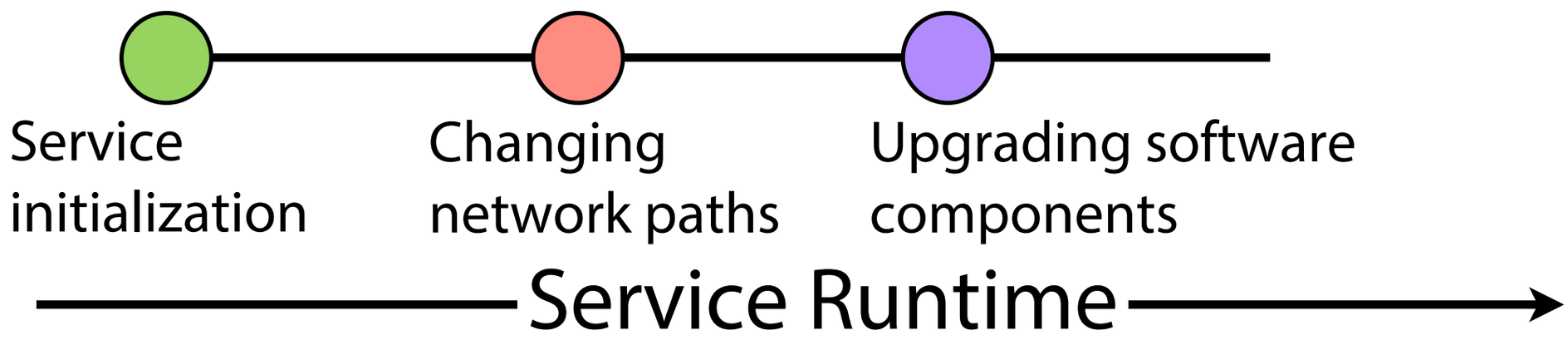
RepAudit [OOPSLA'17]



Our Solution: RepAudit

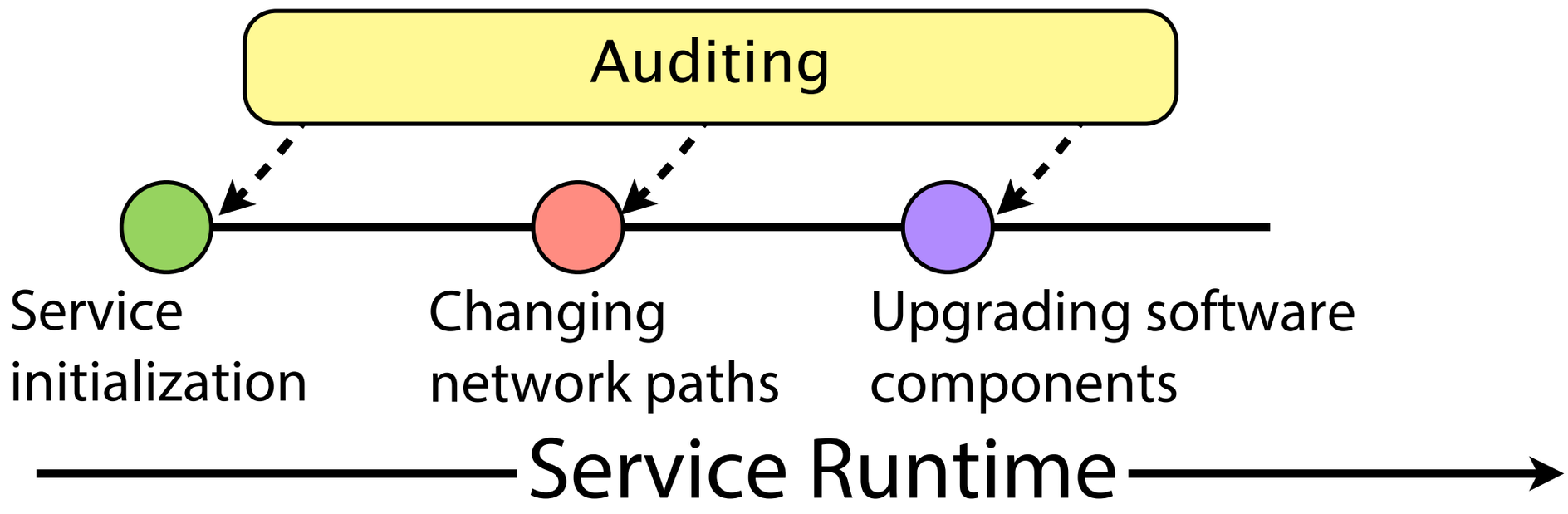
- Hard to express diverse auditing tasks, e.g., identifying risks
  - A new domain-specific auditing language
- Fault graph analysis does not support auditing in runtime

- Have no idea how to fix the cascading failure problem



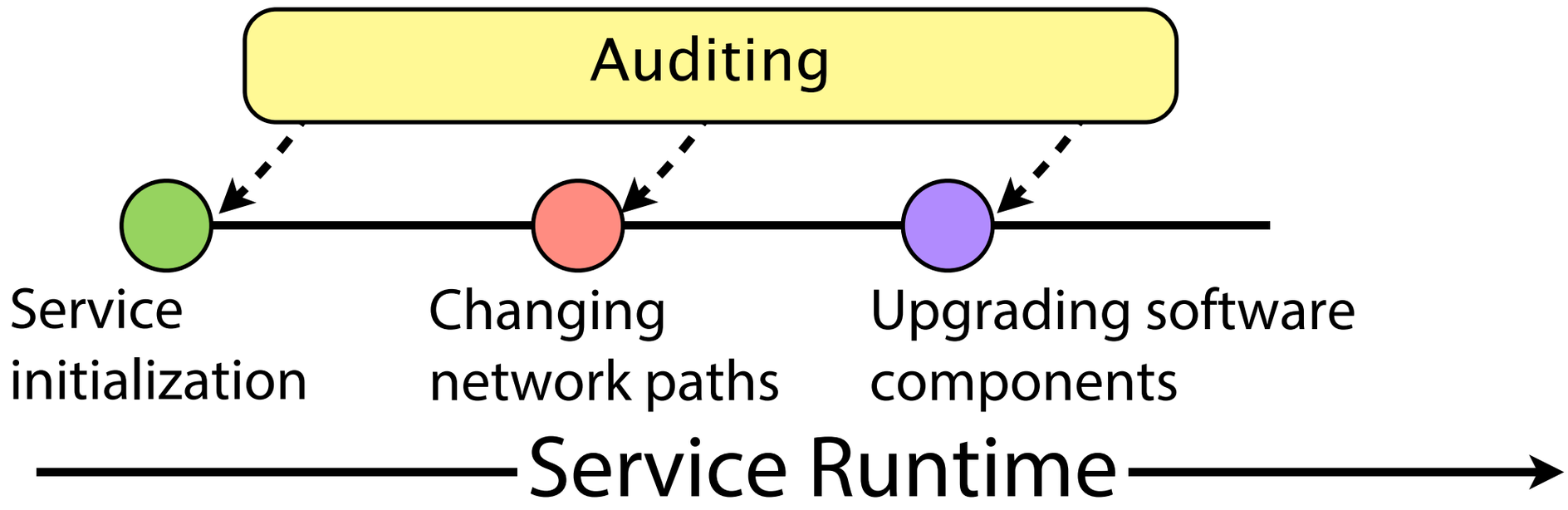
Our Solution: RepAudit

- Hard to express diverse auditing tasks, e.g., identifying risks
  - A new domain-specific auditing language
- Fault graph analysis does not support auditing in runtime
  - Much faster analysis based on various SAT solvers
- Have no idea how to fix the cascading failure problem

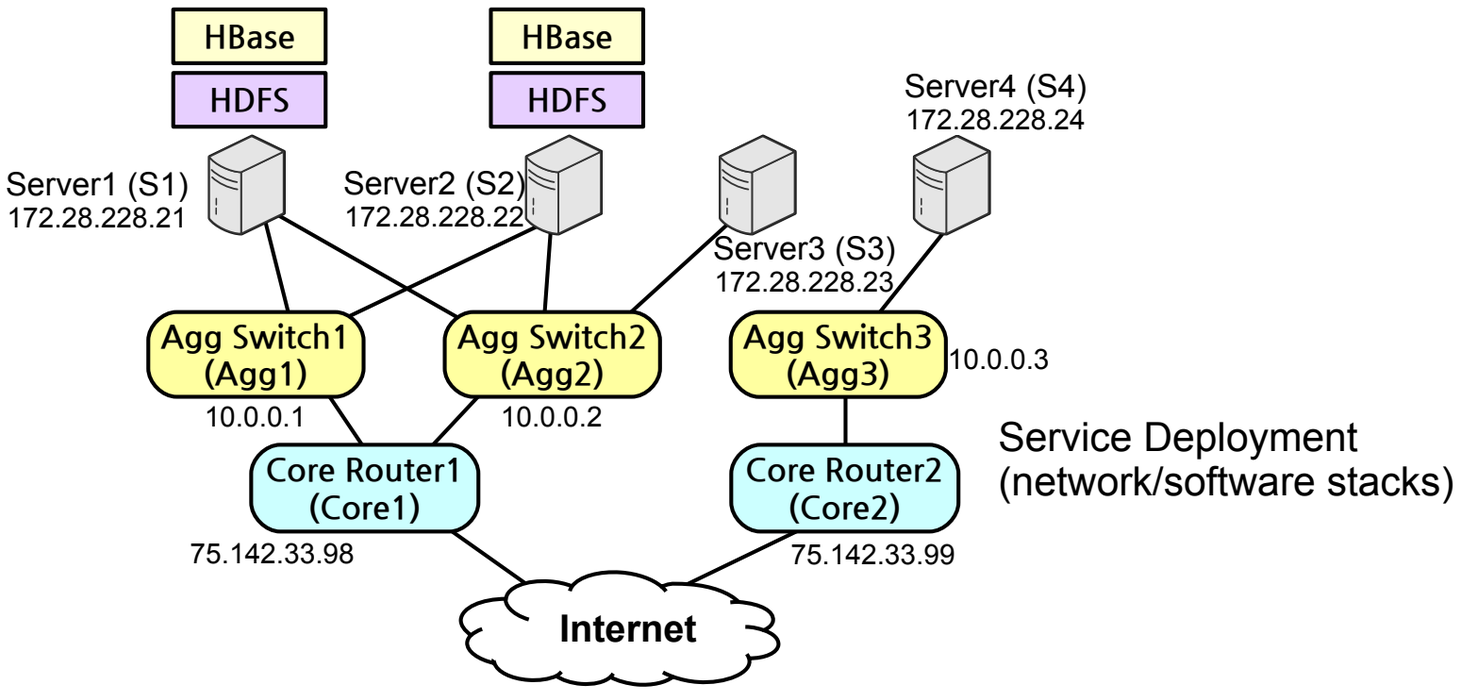


Our Solution: RepAudit

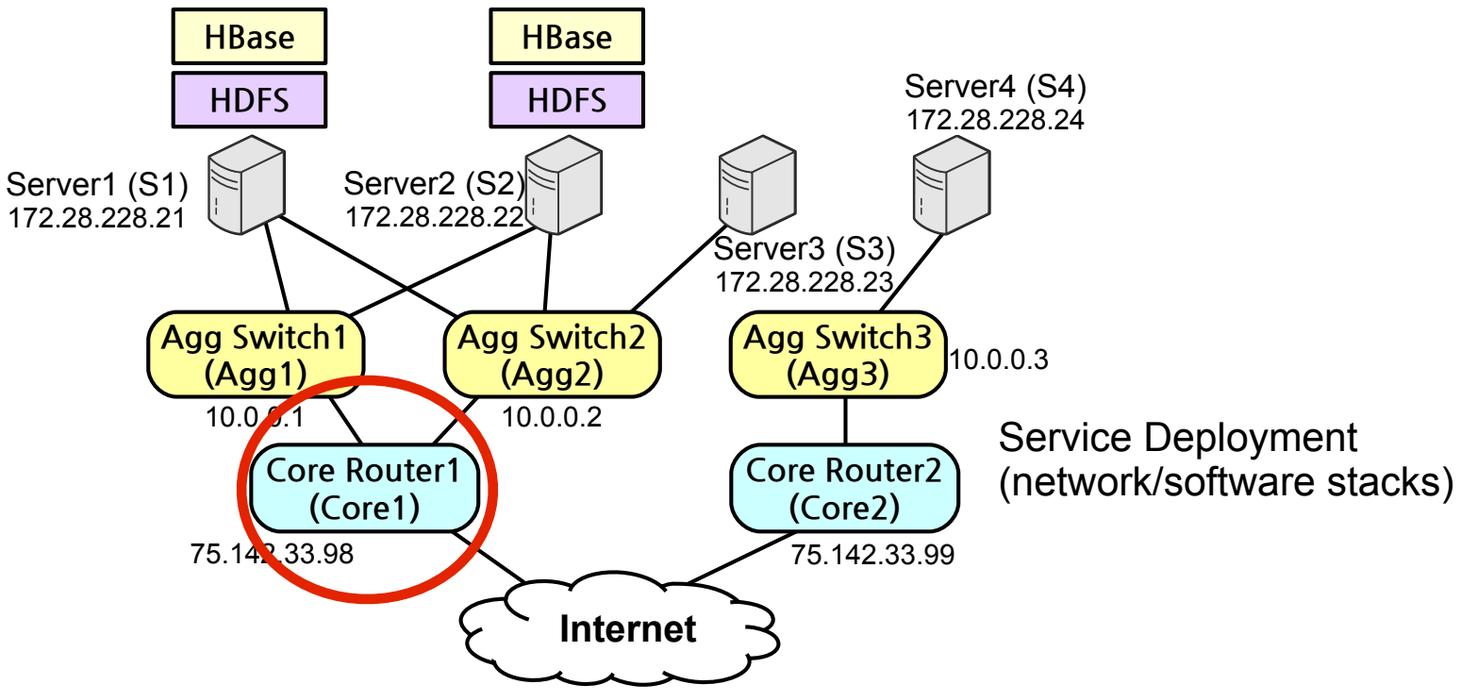
- Hard to express diverse auditing tasks, e.g., identifying risks
  - A new domain-specific auditing language
- Fault graph analysis does not support auditing in runtime
  - Much faster analysis based on various SAT solvers
- Have no idea how to fix the cascading failure problem
  - Automatically generate improvement plans



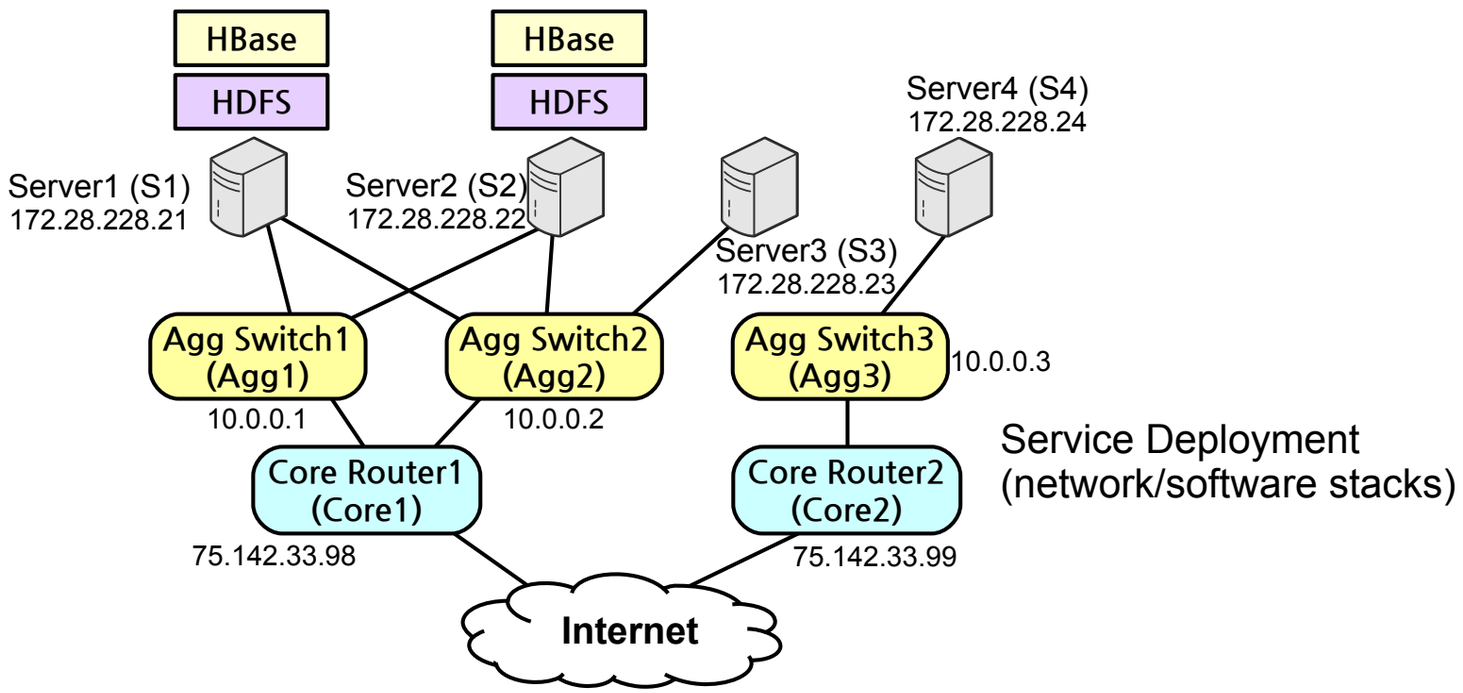
Identifying Unexpected Dependencies



Identifying Unexpected Dependencies



Auditing Engine

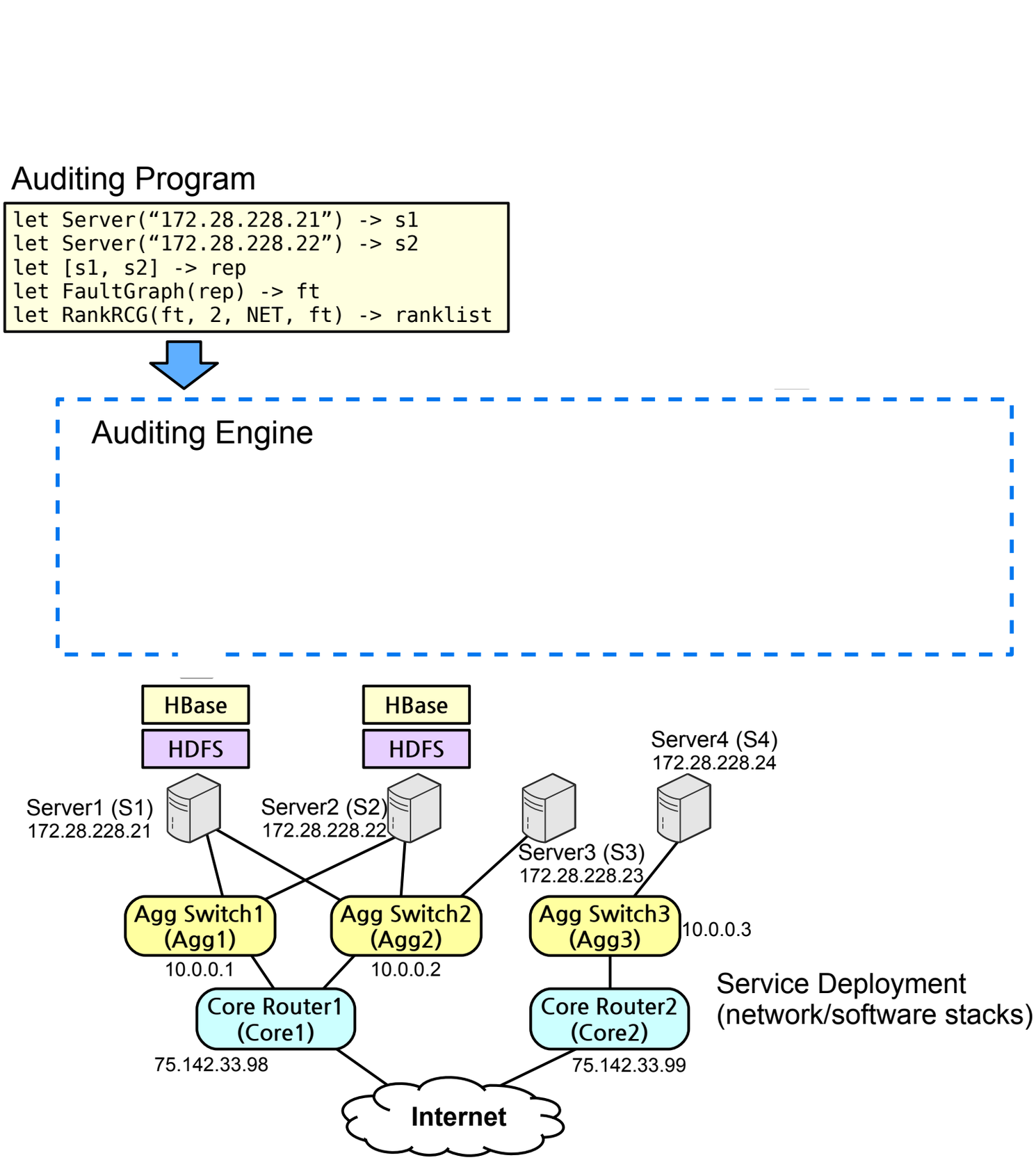


Service Deployment
(network/software stacks)

Auditing Program

```
let Server(“172.28.228.21”) -> s1
let Server(“172.28.228.22”) -> s2
let [s1, s2] -> rep
let FaultGraph(rep) -> ft
let RankRCG(ft, 2, NET, ft) -> ranklist
```

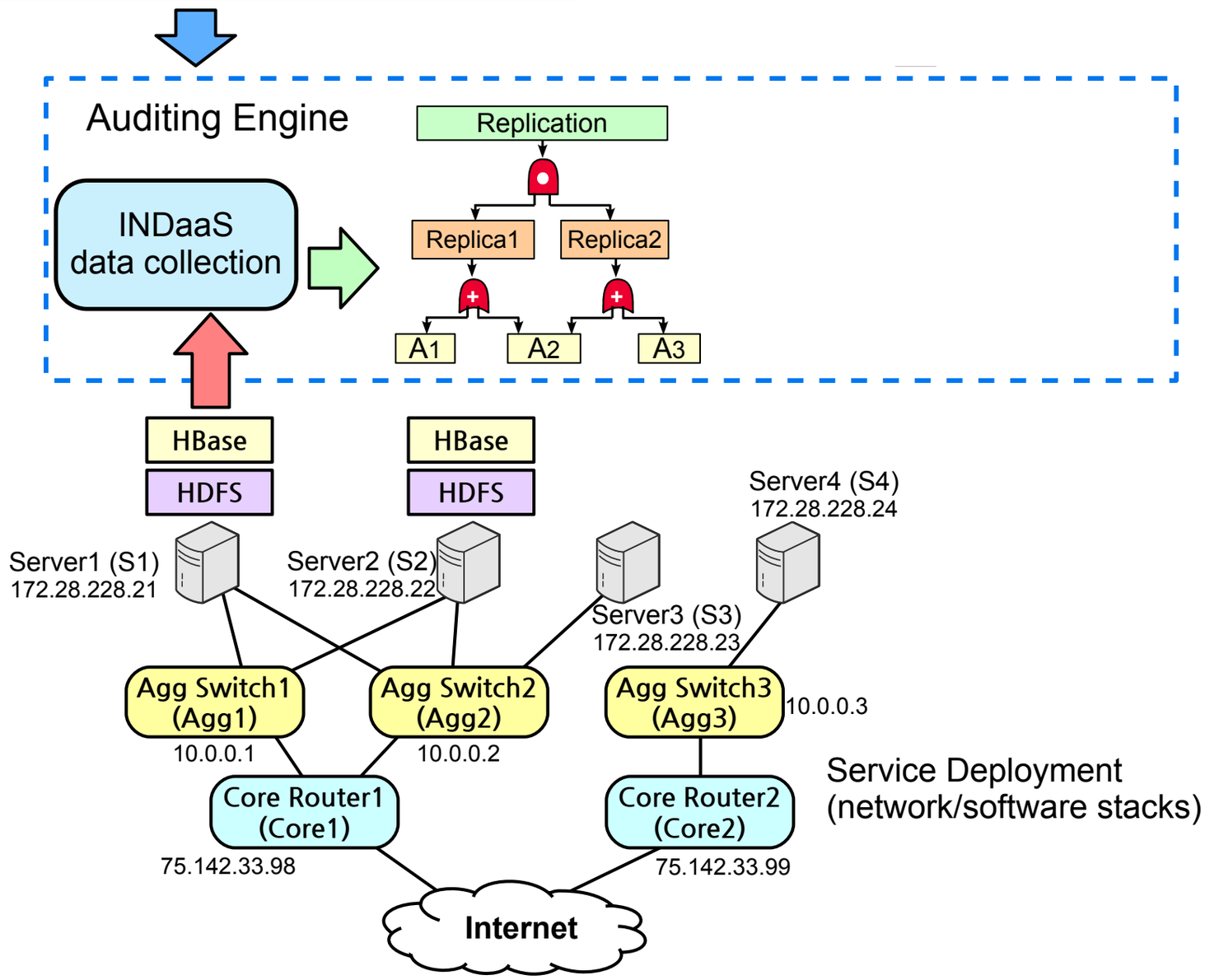
Auditing Engine

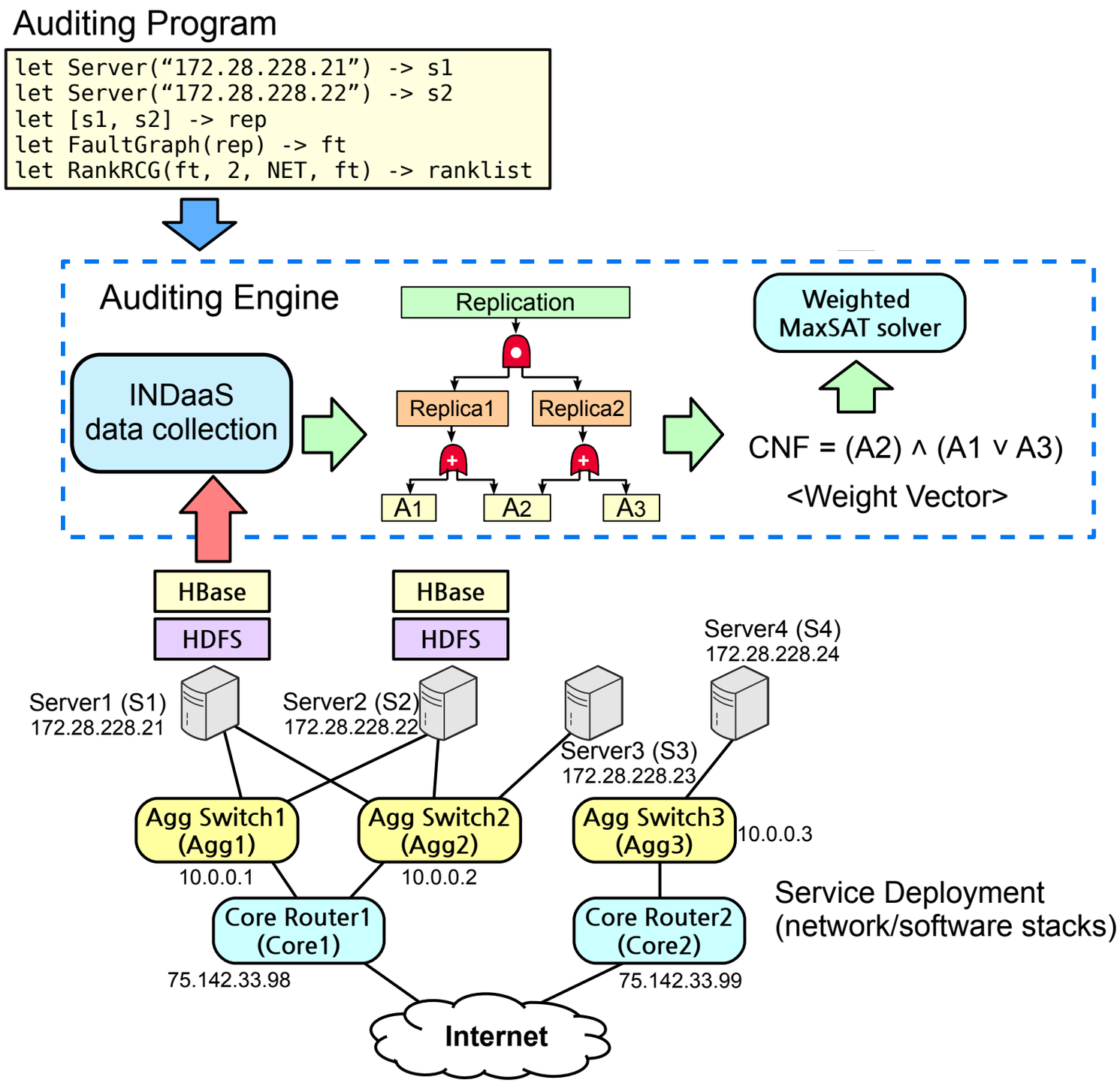


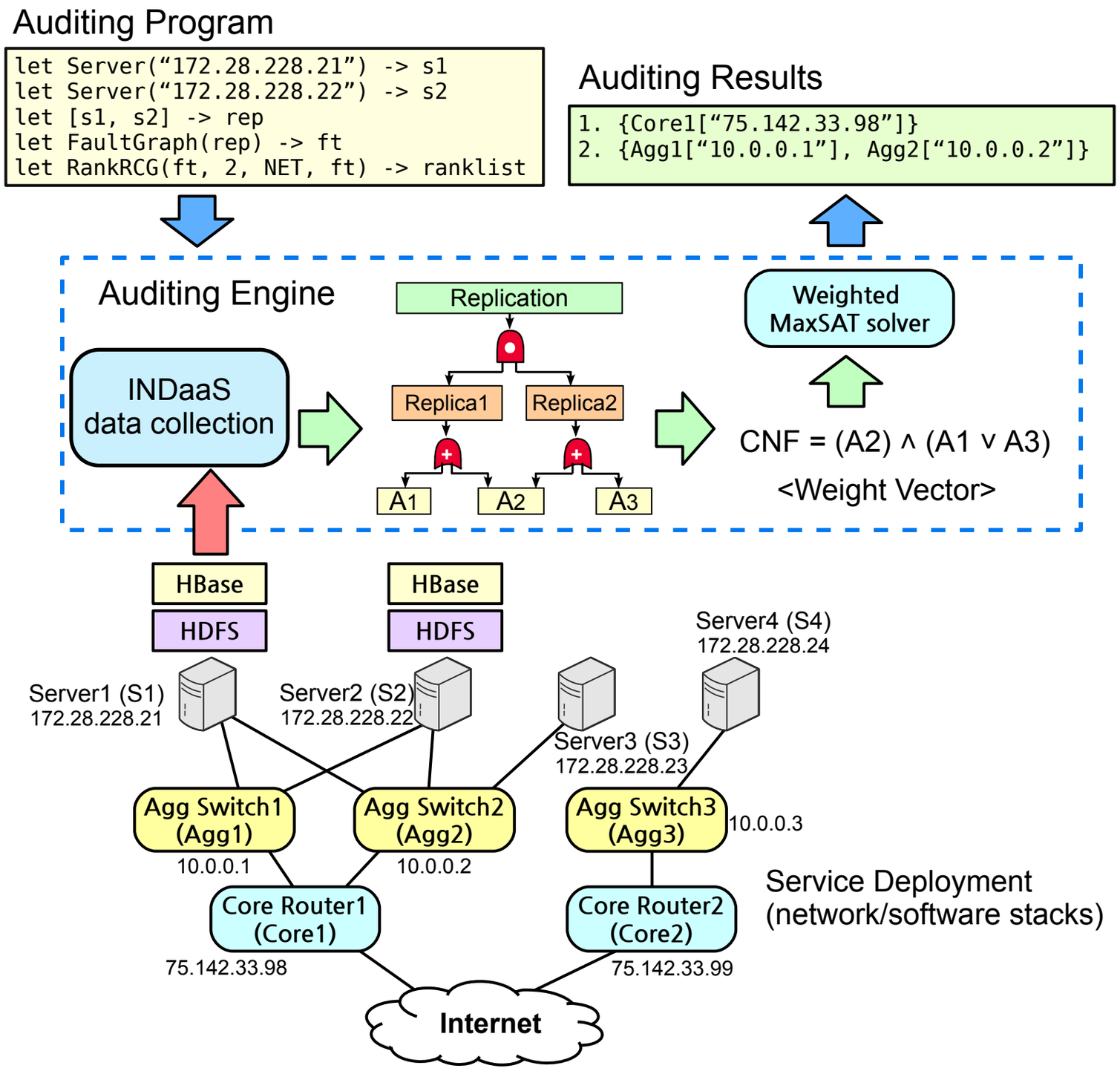
Service Deployment
(network/software stacks)

Auditing Program

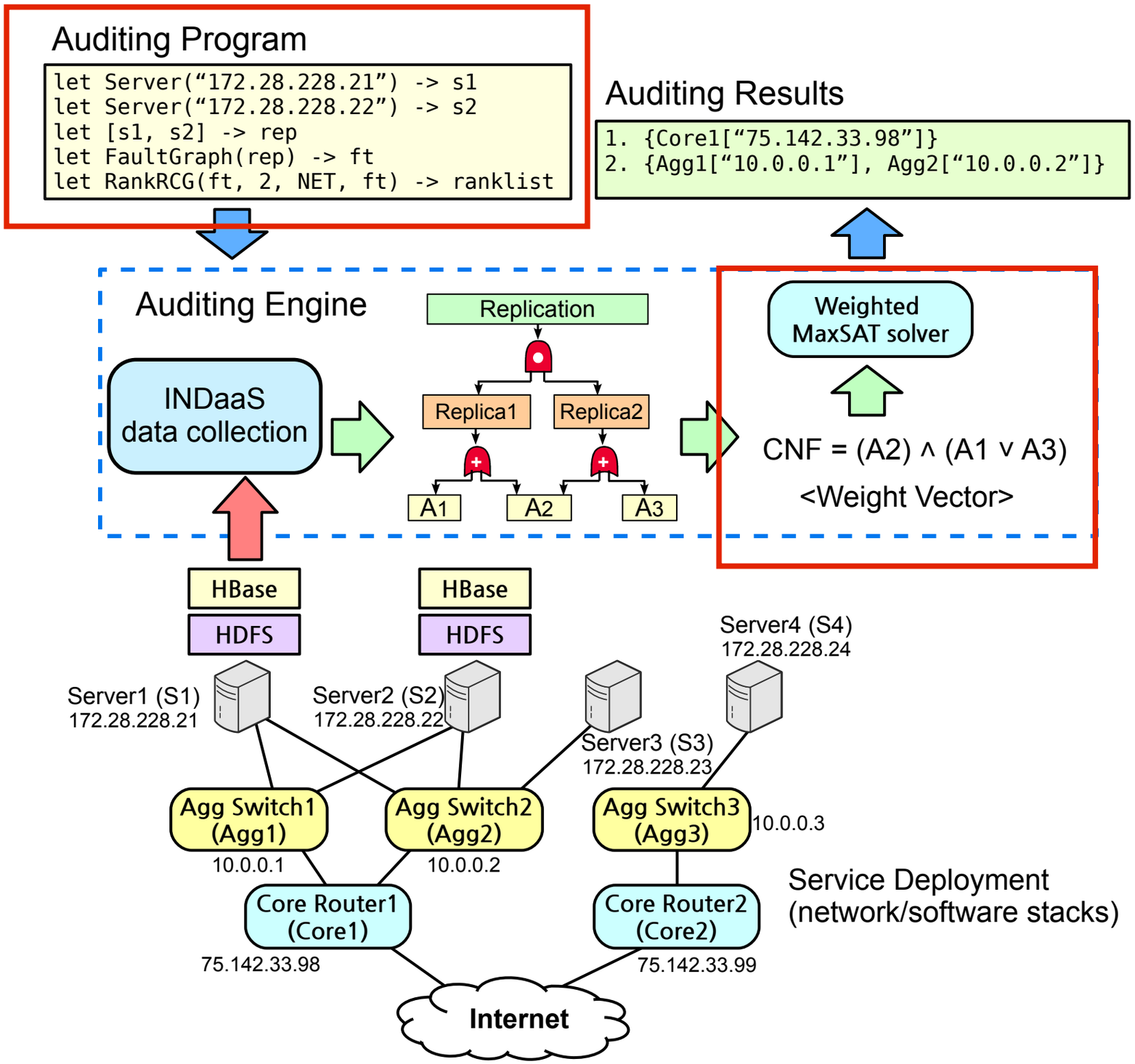
```
let Server(“172.28.228.21”) -> s1
let Server(“172.28.228.22”) -> s2
let [s1, s2] -> rep
let FaultGraph(rep) -> ft
let RankRCG(ft, 2, NET, ft) -> ranklist
```



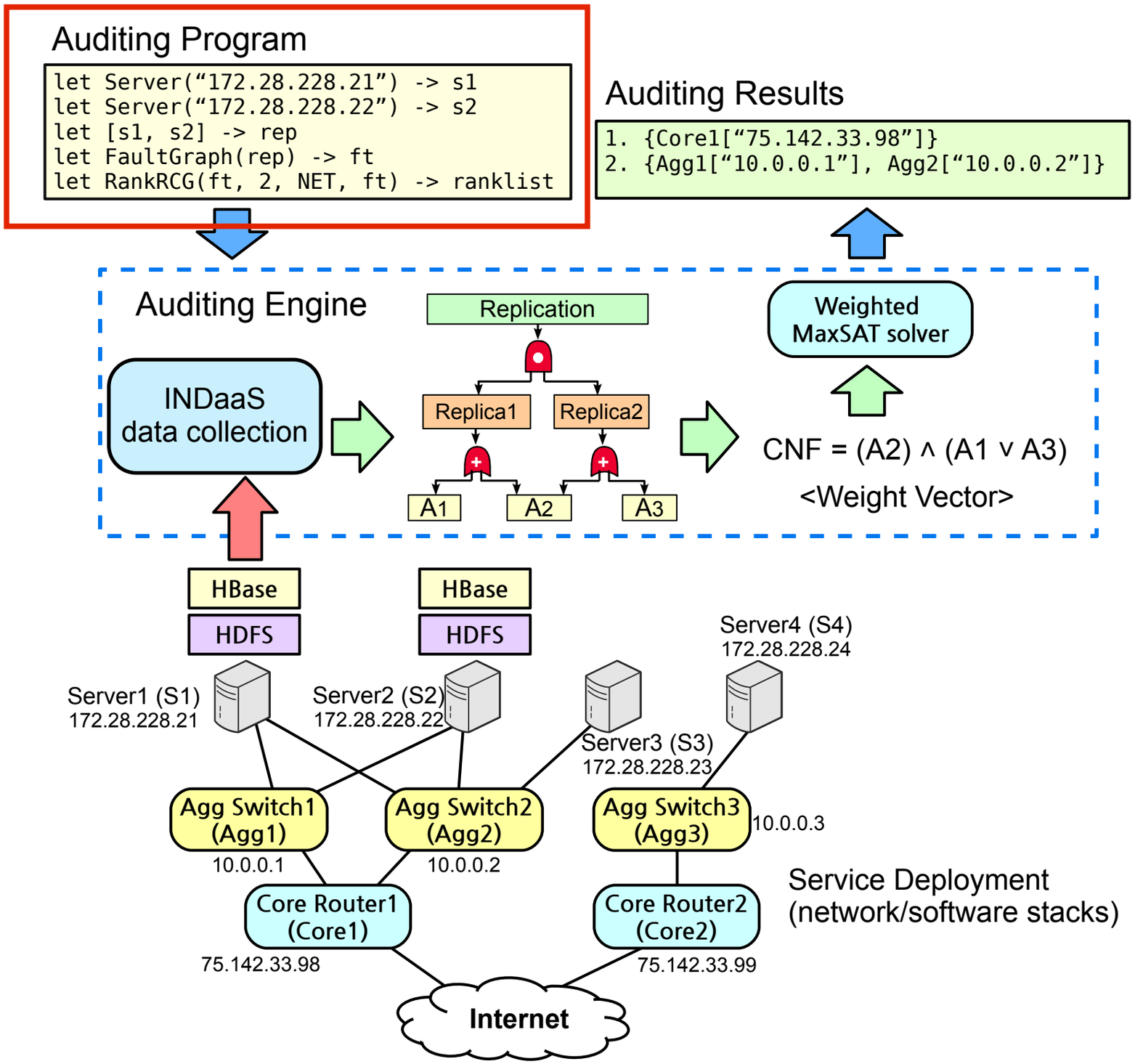

Auditing Program
let Server("172.28.228.21") -> s1
let Server("172.28.228.22") -> s2
let [s1, s2] -> rep
let FaultGraph(rep) -> ft
let RankRCG(ft, 2, NET, ft) -> ranklist
Auditing Engine
INDaaS
data collection
Replication
Replica1
Replica2
A1
A2
A3
Weighted
MaxSAT solver
CNF = (A2) ∧ (A1 ∨ A3)
<Weight Vector>
HBase
HDFS
HBase
HDFS
Server1 (S1)
172.28.228.21
Server2 (S2)
172.28.228.22
Server3 (S3)
172.28.228.23
Server4 (S4)
172.28.228.24
Agg Switch1
(Agg1)
10.0.0.1
Agg Switch2
(Agg2)
10.0.0.2
Agg Switch3
(Agg3)
10.0.0.3
Core Router1
(Core1)
75.142.33.98
Core Router2
(Core2)
75.142.33.99
Internet
Service Deployment
(network/software stacks)


Auditing Program
let Server("172.28.228.21") -> s1
let Server("172.28.228.22") -> s2
let [s1, s2] -> rep
let FaultGraph(rep) -> ft
let RankRCG(ft, 2, NET, ft) -> ranklist
Auditing Results
1. {Core1["75.142.33.98"]}
2. {Agg1["10.0.0.1"], Agg2["10.0.0.2"]}
Auditing Engine
INDaaS
data collection
Replication
Replica1
Replica2
A1
A2
A3
Weighted
MaxSAT solver
CNF = (A2) ∧ (A1 ∨ A3)
<Weight Vector>
HBase
HDFS
HBase
HDFS
Server1 (S1)
172.28.228.21
Server2 (S2)
172.28.228.22
Server3 (S3)
172.28.228.23
Server4 (S4)
172.28.228.24
Agg Switch1
(Agg1)
10.0.0.1
Agg Switch2
(Agg2)
10.0.0.2
Agg Switch3
(Agg3)
10.0.0.3
Core Router1
(Core1)
75.142.33.98
Core Router2
(Core2)
75.142.33.99
Internet
Service Deployment
(network/software stacks)

RepAudit's Contributions



RepAudit's Contributions



Auditing Language

S	::=	$\texttt{let}\ e \rightarrow g$	Assignment
	|	$\texttt{print}(e)$	Output
	|	$S_1; S_2 \mid \texttt{if}(e)\{S_1\}\ \texttt{else}\{S_2\} \mid \texttt{while}(e)\{S\}$	

(a) Statements of RAL.

e	::=	$g \mid c \mid l\langle e \rangle \mid q \mid e_1\ op\ e_2$	Expression
c	::=	$i \mid str$	Real number or string
$l\langle e \rangle$	::=	$\texttt{nil} \mid [e_1, ..., e_n]$	List
op	::=	$< \mid \leq \mid = \mid != \mid > \mid \geq$	Operator
q	::=	$\texttt{Server}(e)$	Initializing server node
	|	$\texttt{Switch}(e)$	Initializing switch node
	|	$\texttt{FaultGraph}(e)$	Generating fault graph
	|	$\texttt{RankRCG}(e_1, e_2, m, t)$	Ranking RCGs
	|	$\texttt{RankNode}(e, m, t)$	Ranking devices
	|	$\texttt{FailProb}(e, t)$	Failure probability
	|	$\texttt{RecRep}(e_1, e_2, m)$	Recommendation
	|	...	
m	::=	$\texttt{SIZE} \mid \texttt{PROB}$	Ranking metric
t	::=	$\texttt{NET} \mid \texttt{SoftW} \mid \texttt{HardW}$	Dependency types

(b) Expressions of RAL.

Auditing Language

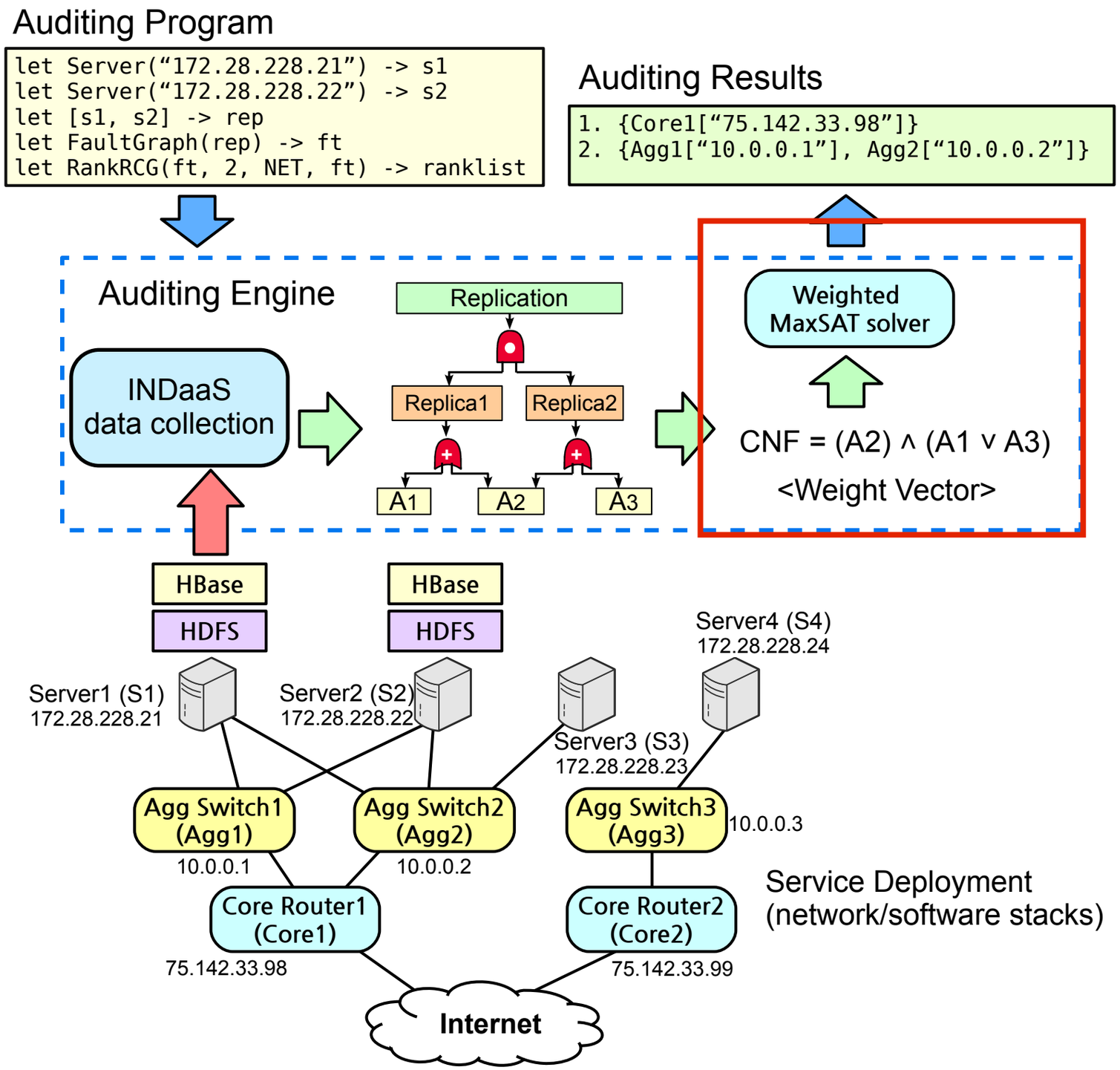
S	::=	`let` $e \rightarrow g$	Assignment
	|	`print`(e)	Output
	|	$S_1;S_2$ | `if`$(e)\{S_1\}$ `else`$\{S_2\}$ | `while`$(e)\{S\}$	

(a) Statements of RAL.

e	::=	$g \mid c \mid l\langle e\rangle \mid q \mid e_1\ op\ e_2$	Expression
c	::=	$i \mid str$	Real number or string
$l\langle e\rangle$	::=	`nil` $\mid [e_1,...,e_n]$	List
op	::=	$< \mid \leq \mid = \mid != \mid > \mid \geq$	Operator
q	::=	`Server`(e)	Initializing server node
	|	`Switch`(e)	Initializing switch node
	|	`FaultGraph`(e)	Generating fault graph
	|	`RankRCG`(e_1,e_2,m,t)	Ranking RCGs
	|	`RankNode`(e,m,t)	Ranking devices
	|	`FailProb`(e,t)	Failure probability
	|	`RecRep`(e_1,e_2,m)	Recommendation
	|	...	
m	::=	`SIZE` | `PROB`	Ranking metric
t	::=	`NET` | `SoftW` | `HardW`	Dependency types

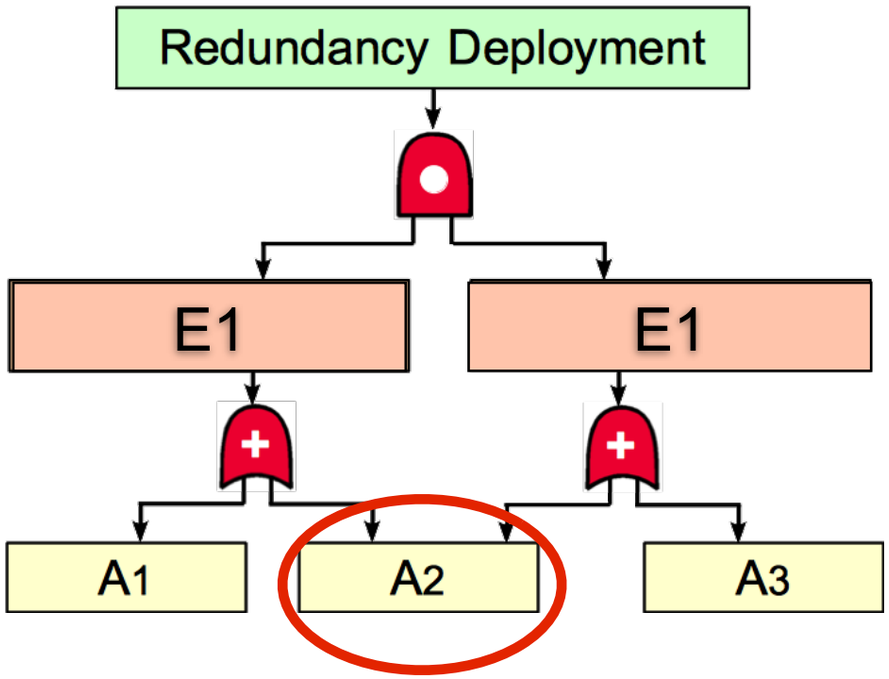
(b) Expressions of RAL.

Identifying Shared Dependencies



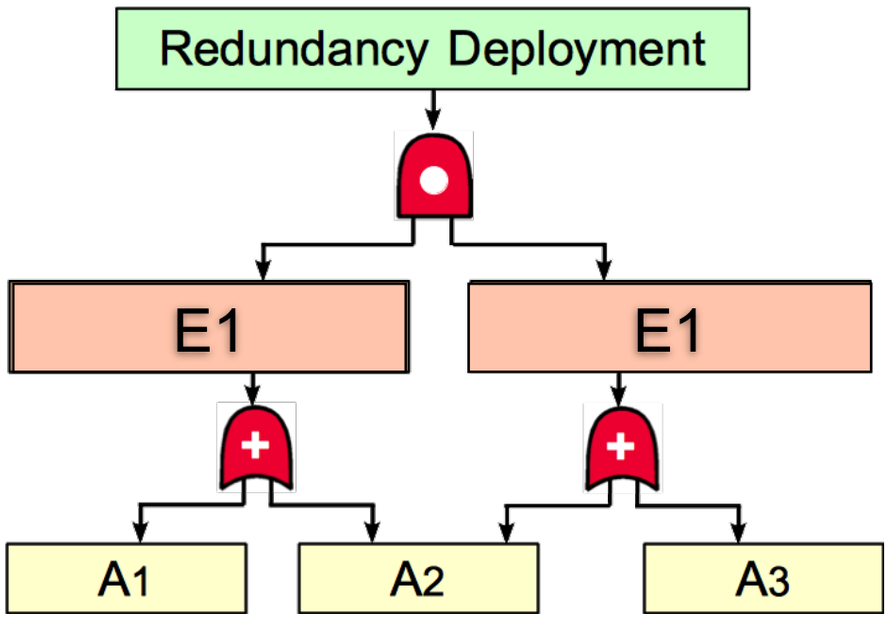
Risk Groups in Fault Graphs

Risk Groups in Fault Graphs



A risk group means a set of leaf nodes whose simultaneous failures lead to the failure of root node.

Risk Groups in Fault Graphs

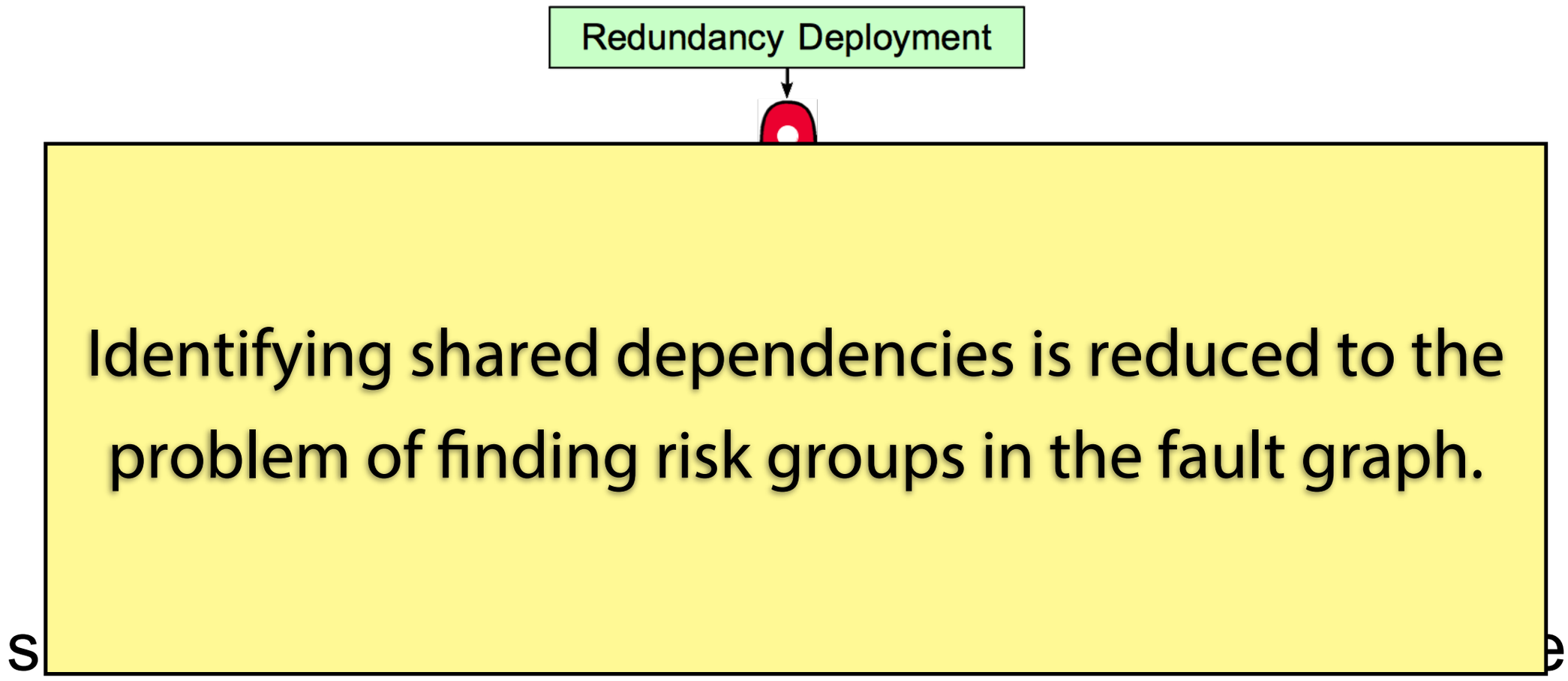


A risk group means a set of leaf nodes whose simultaneous failures lead to the failure of root node.

{A2} and {A1, A3} are risk groups

{A1} or {A3} is not risk group

Risk Groups in Fault Graphs



s e.

{A2} and {A1, A3} are risk groups

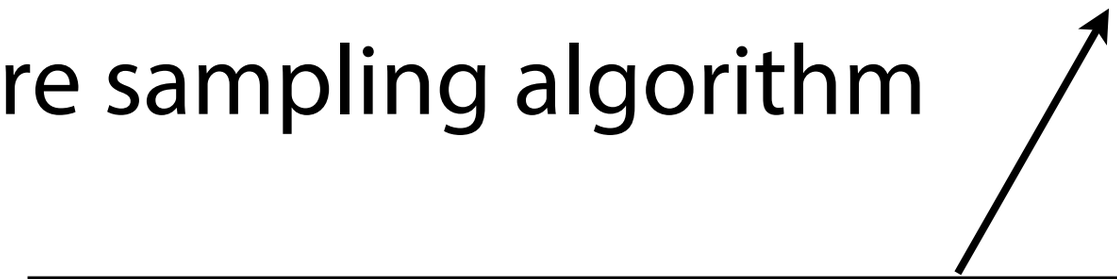
{A1} or {A3} is not risk group

State-of-the-Art Risk Group Analysis

- State-of-the-art risk group detection efforts:
  - Deterministic minimal cut set algorithm
  - Failure sampling algorithm

State-of-the-Art Risk Group Analysis

- State-of-the-art risk group detection efforts:
  - Deterministic minimal cut set algorithm
  - Failure sampling algorithm



Pros: 100% accurate results
Cons: Exponential-time complexity

State-of-the-Art Risk Group Analysis

- State-of-the-art risk group detection efforts:
  - Deterministic minimal cut set algorithm
  - Failure sampling algorithm

Pros: 100% accurate results
Cons: Exponential-time complexity

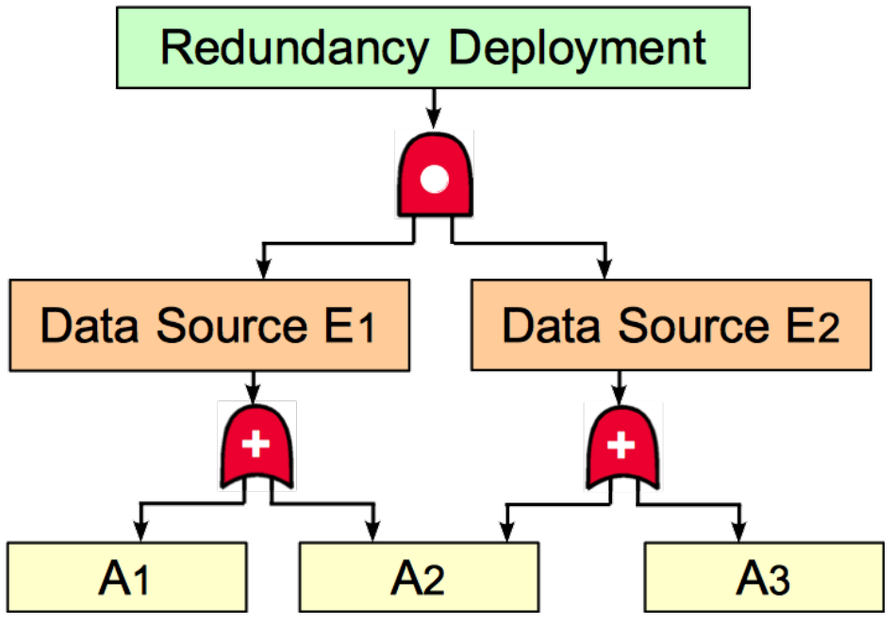
Pros: Efficient auditing approach
Cons: Accuracy is quite low in large system

State-of-the-Art Risk Group Analysis

- State-of-the-art risk group detection efforts:
  - Deterministic minimal cut set algorithm
  - Pros: Efficient auditing approach
    Cons: Accuracy is quite low in large system

We want to achieve both efficiency and accuracy in large-scale system auditing

Our Insight

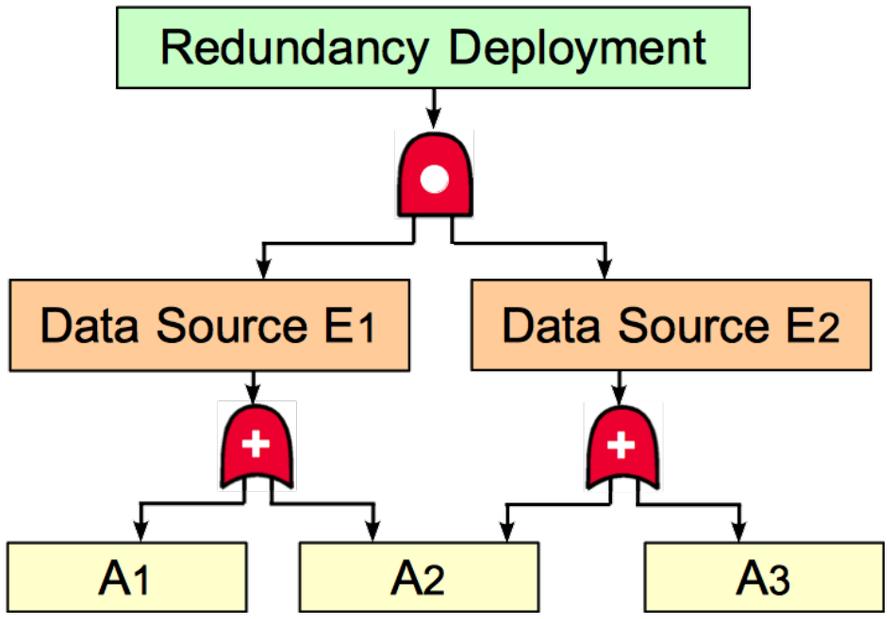


Boolean formula

$= E_1 \wedge E_2$

$= (A_1 \vee A_2) \wedge (A_2 \vee A_3)$

Our Insight



Boolean formula

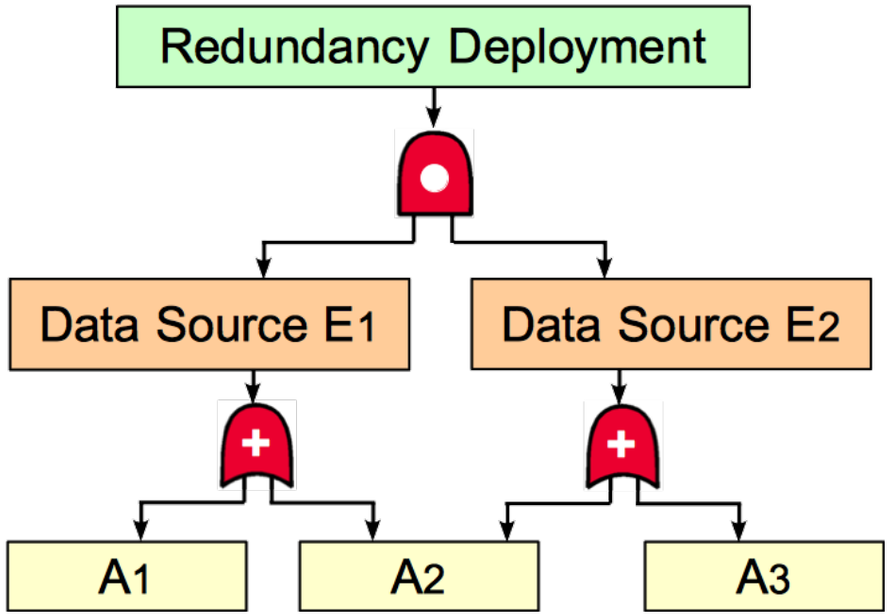
$= E_1 \wedge E_2$

$= (A_1 \vee A_2) \wedge (A_2 \vee A_3)$

- Extracting risk groups can be reduced to the problem of extracting satisfying assignments from boolean formula

- E.g., {A1=0, A2=1, A3=0} represents a risk group

Our Insight



Boolean formula

$= E_1 \wedge E_2$

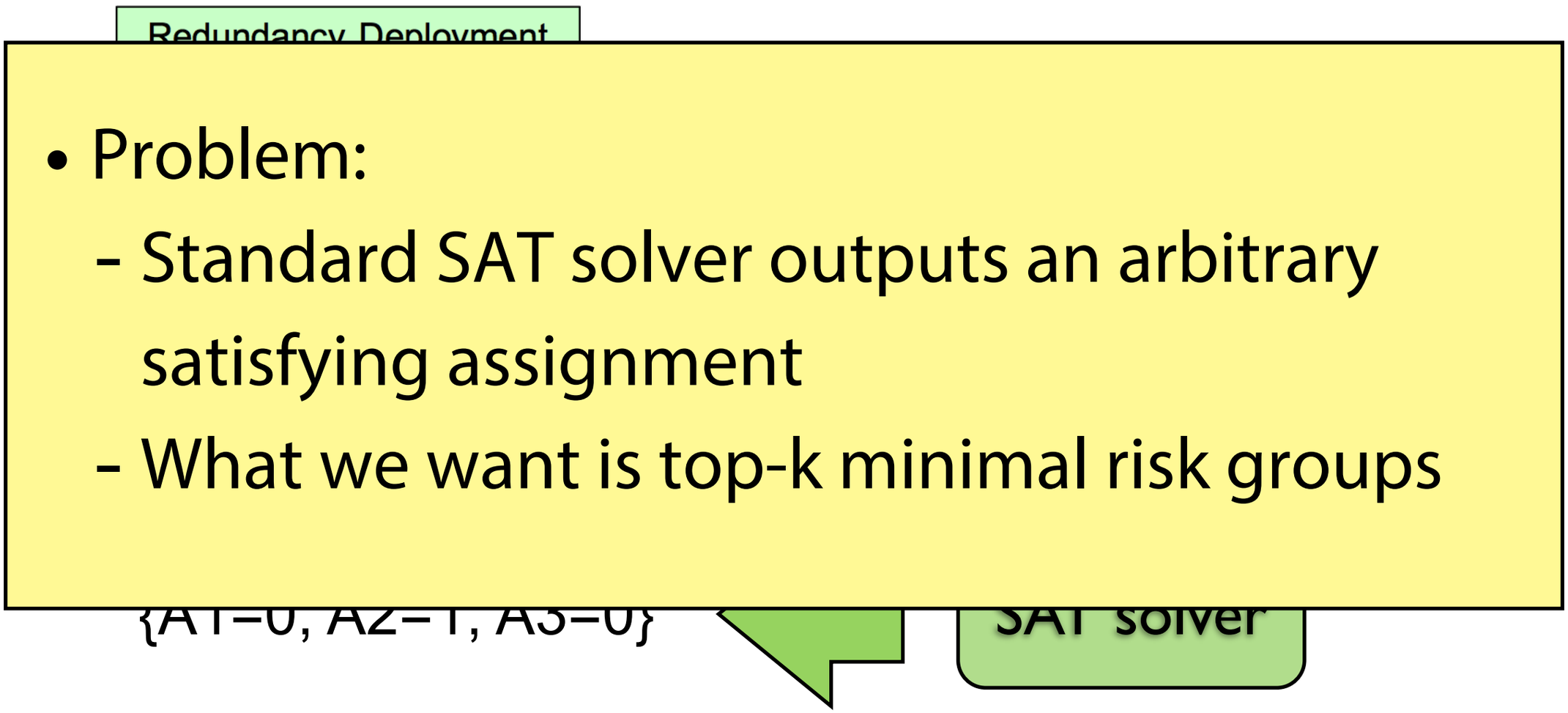
$= (A_1 \vee A_2) \wedge (A_2 \vee A_3)$

SAT solver

{A1=0, A2=1, A3=0}

Our Insight

- Problem:
  - Standard SAT solver outputs an arbitrary satisfying assignment
  - What we want is top-k minimal risk groups



Discovering Risk Groups

Discovering Risk Groups

- Using weighted MaxSAT solver
  - Satisfiable assignment with the least weights
  - Obtain the least $C = \sum c_i \cdot w_i$
  - Very fast with 100% accuracy

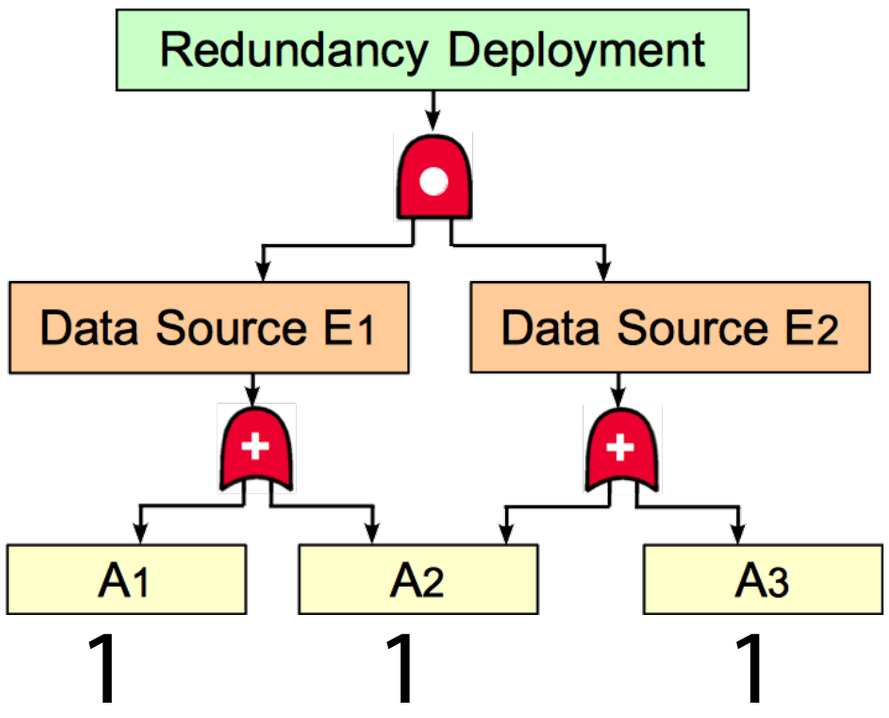
Discovering Risk Groups

- Using weighted MaxSAT solver
  - Satisfiable assignment with the least weights
  - Obtain the least $C = \sum c_i \cdot w_i$
  - Very fast with 100% accuracy

We set the values of all the leaf nodes as 1

Discovering Risk Groups

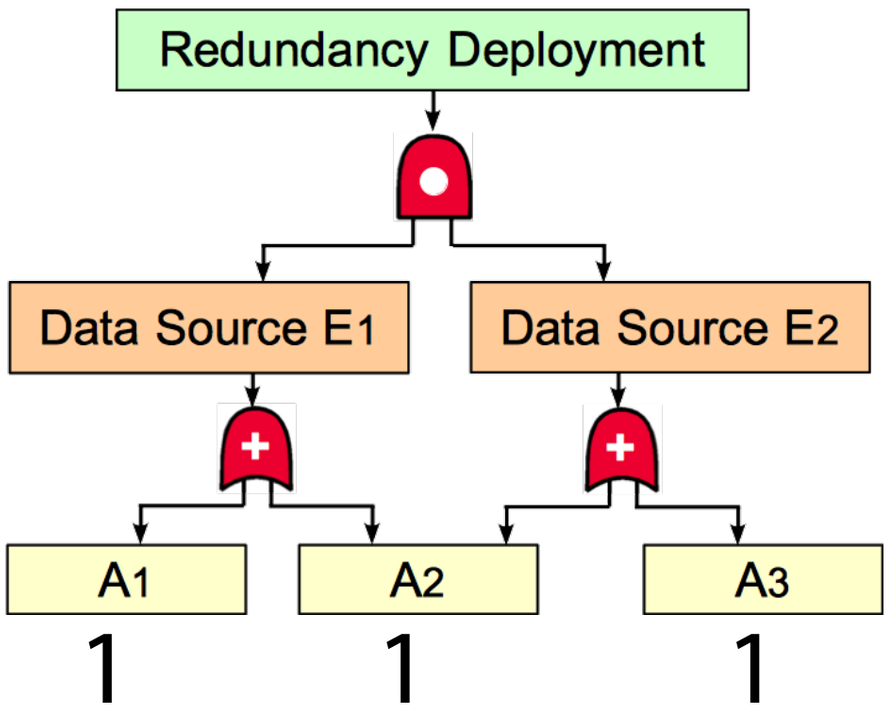
- Using weighted MaxSAT solver
  - Satisfiable assignment with the least weights
  - Obtain the least $C = \sum c_i \cdot w_i$
  - Very fast with 100% accuracy



A1	A2	A3	weight
1	0	0	
0	1	0	1
0	0	1	
1	1	0	2
1	0	1	2
0	1	1	2
0	0	0	
1	1	1	3

Discovering Risk Groups

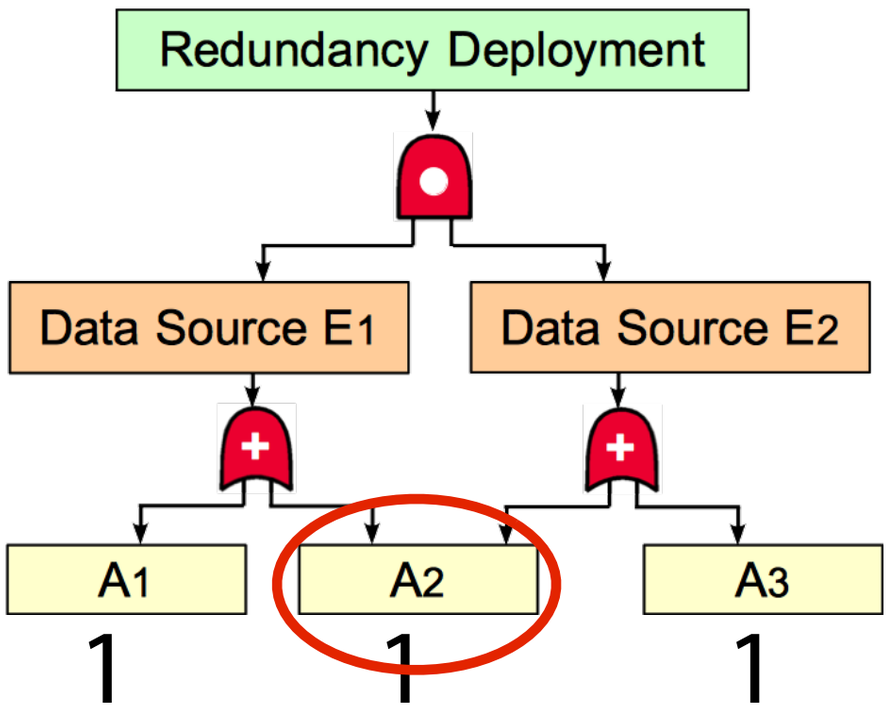
- Using weighted MaxSAT solver
  - Satisfiable assignment with the least weights
  - Obtain the least $C = \sum c_i \cdot w_i$
  - Very fast with 100% accuracy



A1	A2	A3	weight
1	0	0	
0	1	0	1
0	0	1	
1	1	0	2
1	0	1	2
0	1	1	2
0	0	0	
1	1	1	3

Discovering Risk Groups

- Using weighted MaxSAT solver
  - Satisfiable assignment with the least weights
  - Obtain the least $C = \sum c_i \cdot w_i$
  - Very fast with 100% accuracy



A1	A2	A3	weight
1	0	0	
0	1	0	1
0	0	1	
1	1	0	2
1	0	1	2
0	1	1	2
0	0	0	
1	1	1	3

Discovering Risk Groups

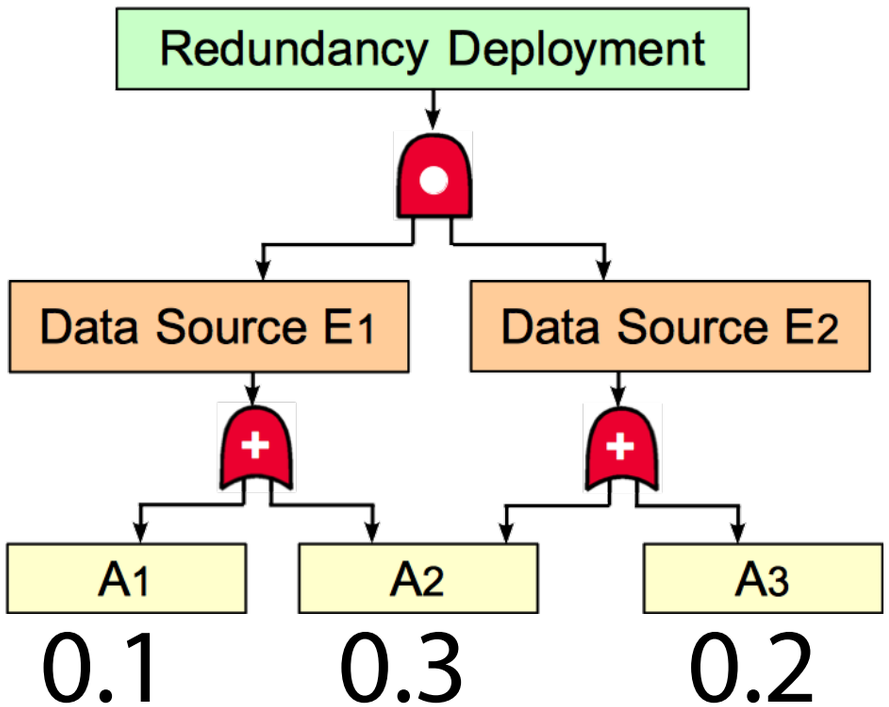
- Find out the top-k critical risk groups
  - Use a $\wedge$ to connect the current formula and negation of the resulting assignment

Discovering Risk Groups

- Find out the top-k critical risk groups
  - Use a $\wedge$ to connect the current formula and negation of the resulting assignment

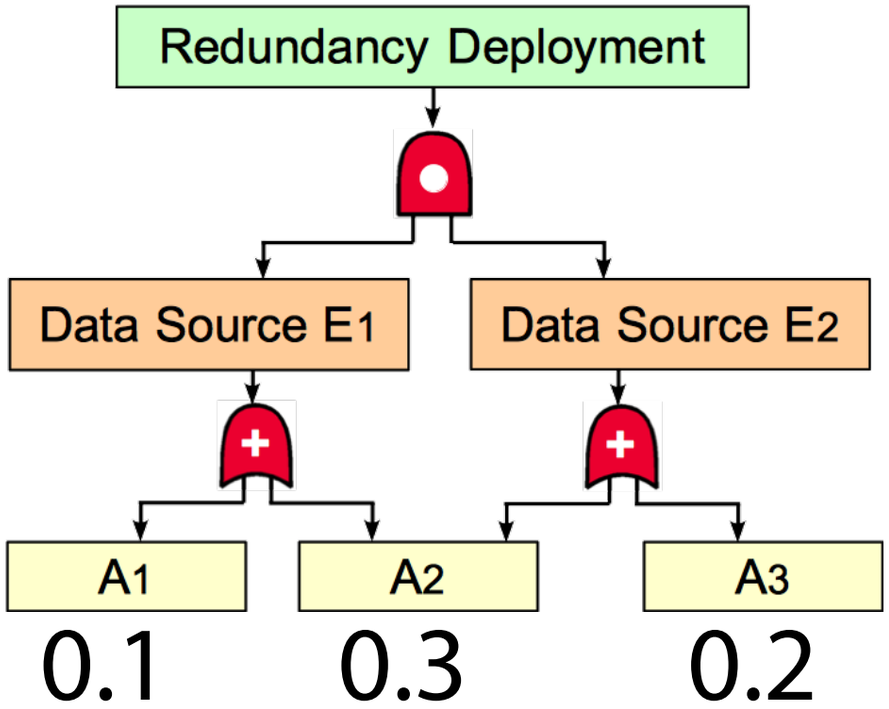
$$(A_1 \vee A_2) \wedge (A_2 \vee A_3) \quad \wedge \neg(\neg A_1 \wedge A_2 \wedge \neg A_3)$$

If we can obtain failure probability of each component, then



A1	A2	A3	weight
1	0	0	
0	1	0	0.3
0	0	1	
1	1	0	0.03
1	0	1	0.02
0	1	1	0.06
0	0	0	
1	1	1	0.006

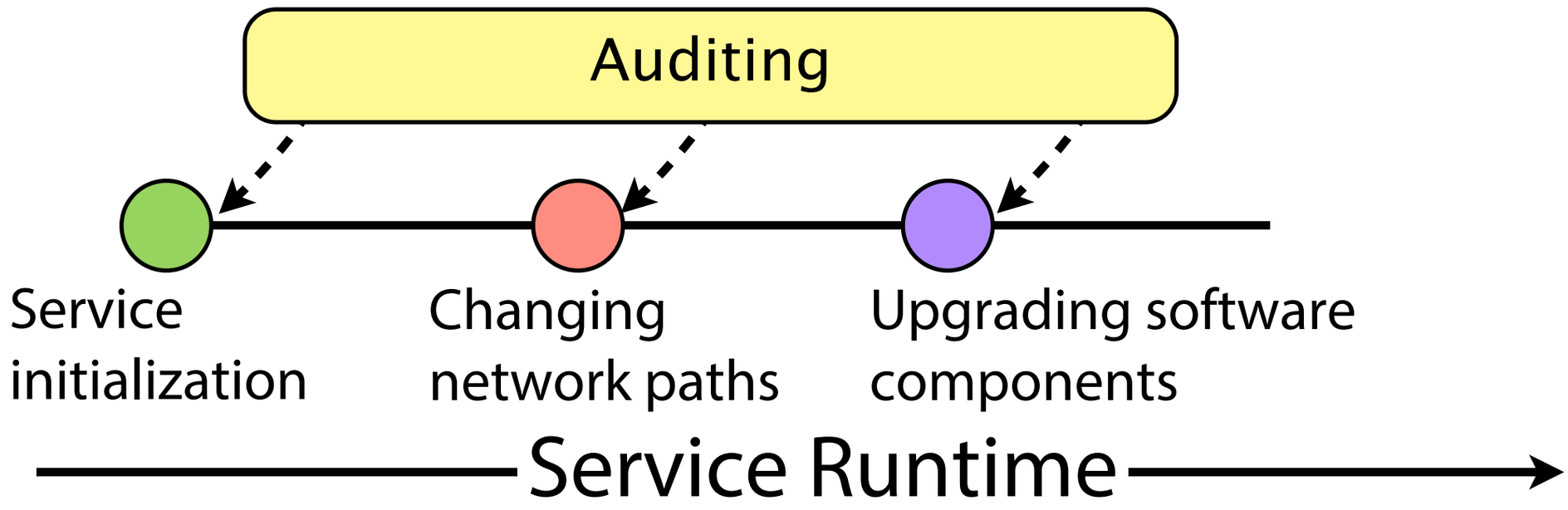
If we can obtain failure probability of each component, then



A1	A2	A3	weight
1	0	0	
0	1	0	0.3
0	0	1	
1	1	0	0.03
1	0	1	0.02
0	1	1	0.06
0	0	0	
1	1	1	0.006

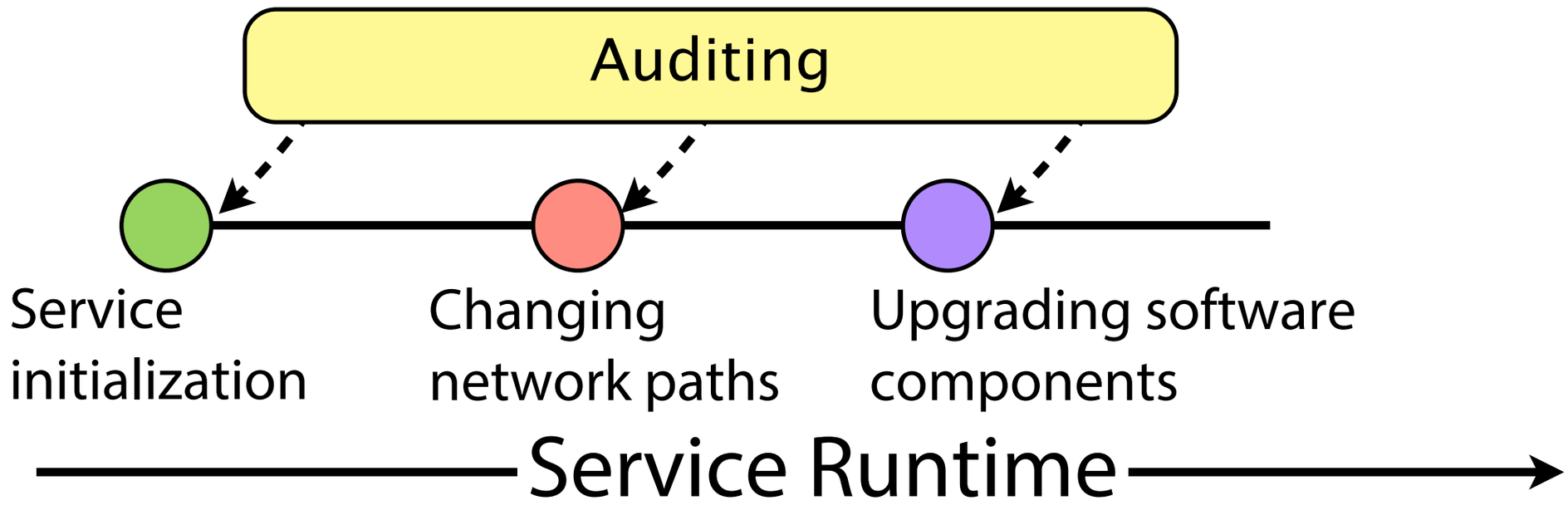
Issues in INDaaS

- Hard to express diverse auditing tasks
  - A new domain-specific auditing language
- Fault graph analysis does not support auditing in runtime
  - Much faster analysis based on SAT solver variants
- Have no idea how to fix the cascading failure problem
  - Automatically generate improvement plans

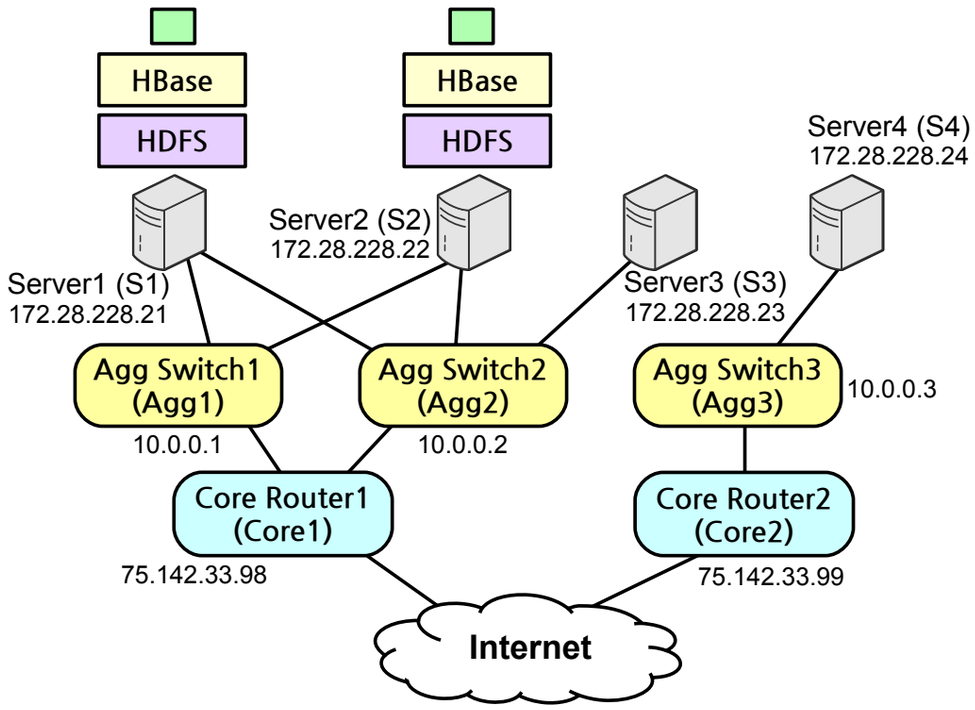


Issues in INDaaS

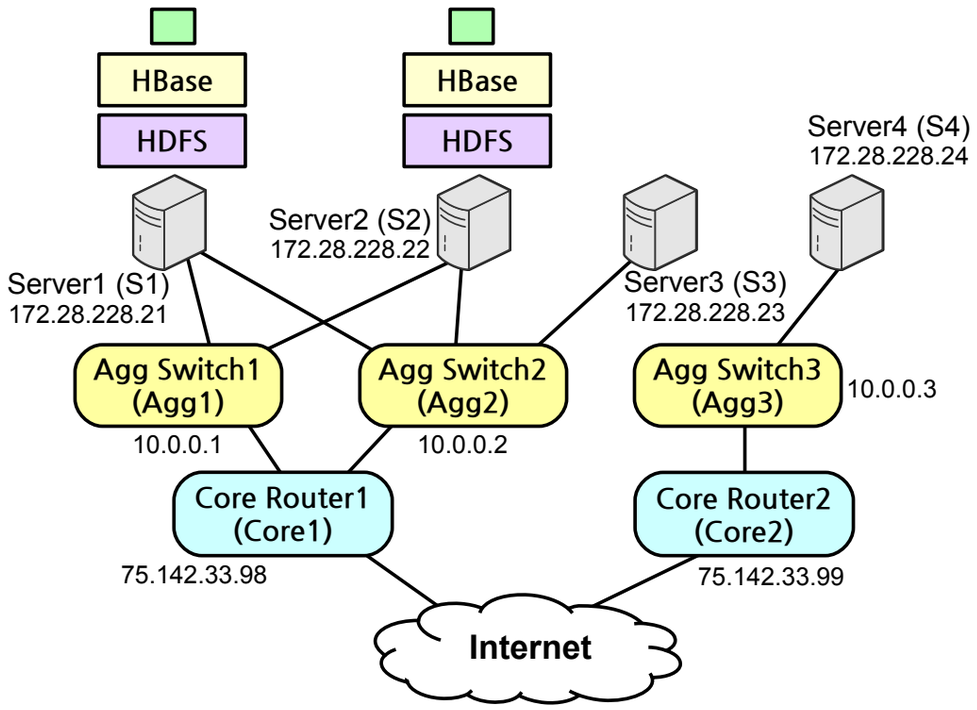
- Hard to express diverse auditing tasks
  - A new domain-specific auditing language
- Fault graph analysis does not support auditing in runtime
  - Much faster analysis based on SAT solver variants
- Have no idea how to fix the cascading failure problem
  - Automatically generate improvement plans



Repair


HBase
HDFS
HBase
HDFS
Server4 (S4)
172.28.228.24
Server2 (S2)
172.28.228.22
Server1 (S1)
172.28.228.21
Server3 (S3)
172.28.228.23
Agg Switch1
(Agg1)
Agg Switch2
(Agg2)
Agg Switch3
(Agg3)
10.0.0.3
10.0.0.1
10.0.0.2
Core Router1
(Core1)
Core Router2
(Core2)
75.142.33.98
75.142.33.99
Internet

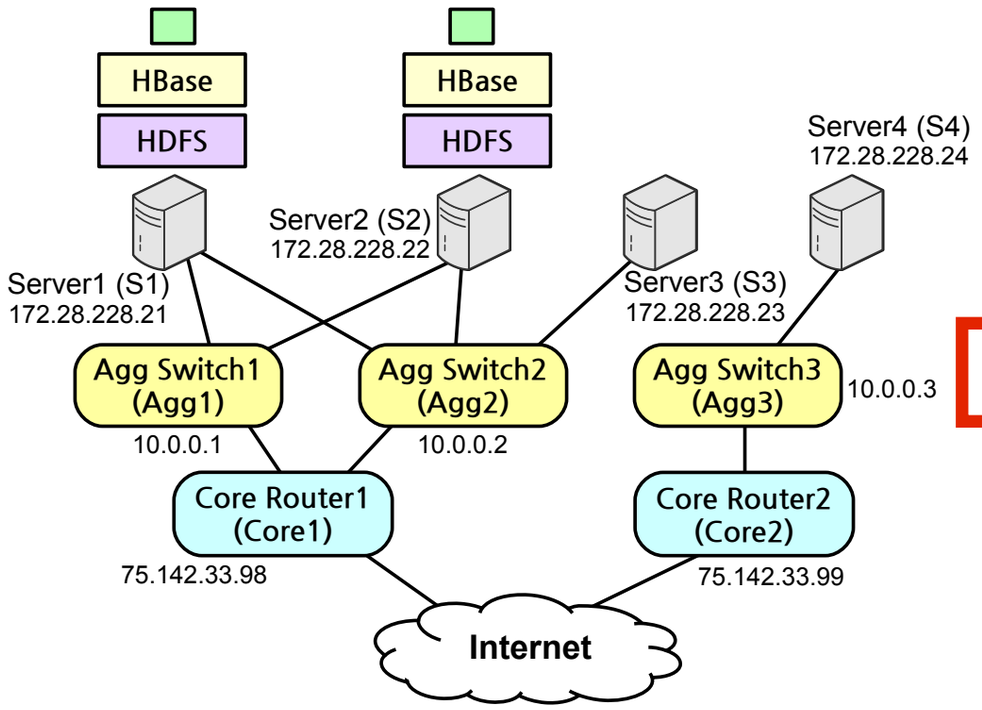
Repair



Specification:

```
$Server -> 172.28.228.21, 172.28.228.22
goal(failProb(ft)<0.08 | ChNode | Agg3)
```

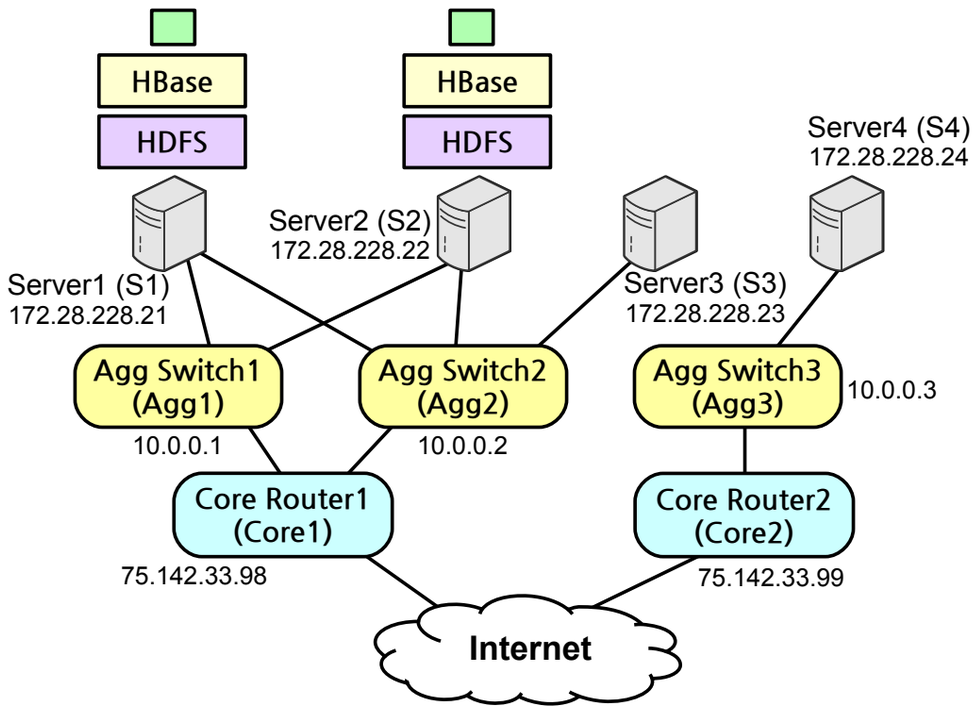
Repair



Specification:

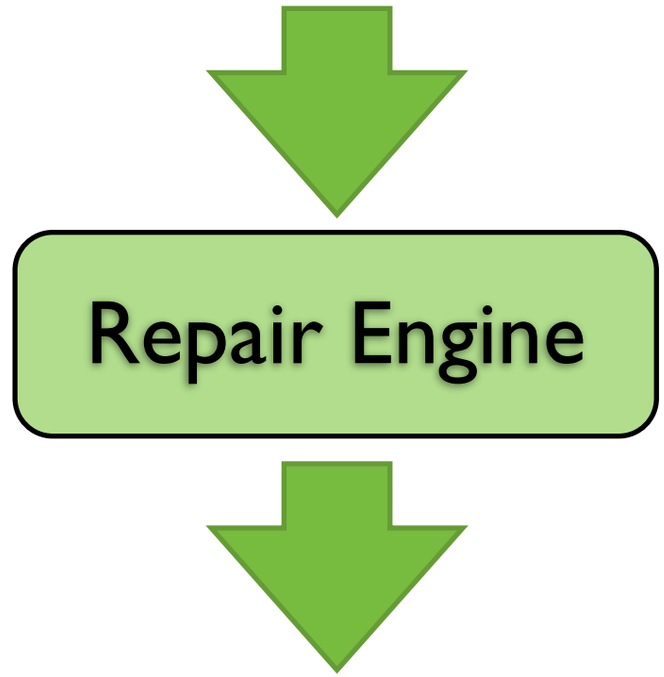
```
$Server -> 172.28.228.21, 172.28.228.22
goal(failProb(ft)<0.08 | ChNode | Agg3)
```

Repair



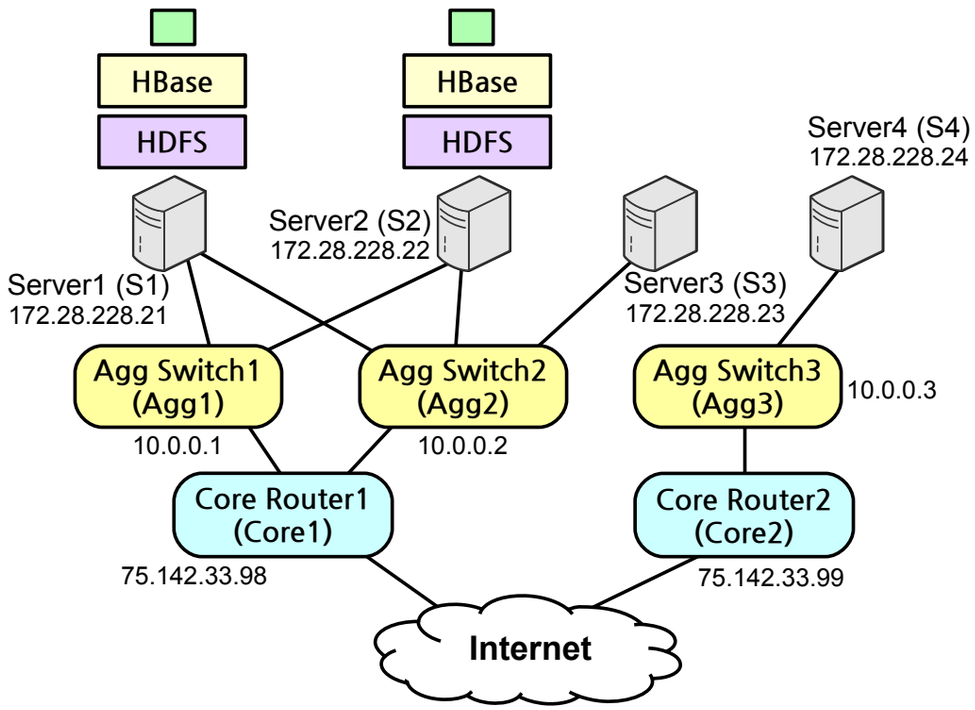
Specification:

```
$Server -> 172.28.228.21, 172.28.228.22
goal(failProb(ft)<0.08 | ChNode | Agg3)
```



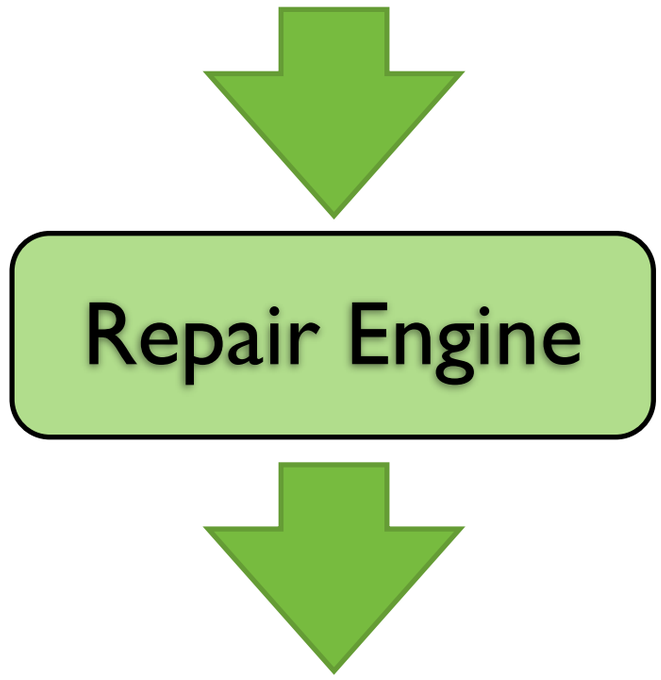

```
Plan 1: Move replica from S1 -> S4
Plan 2: Move replica from S2 -> S4
```

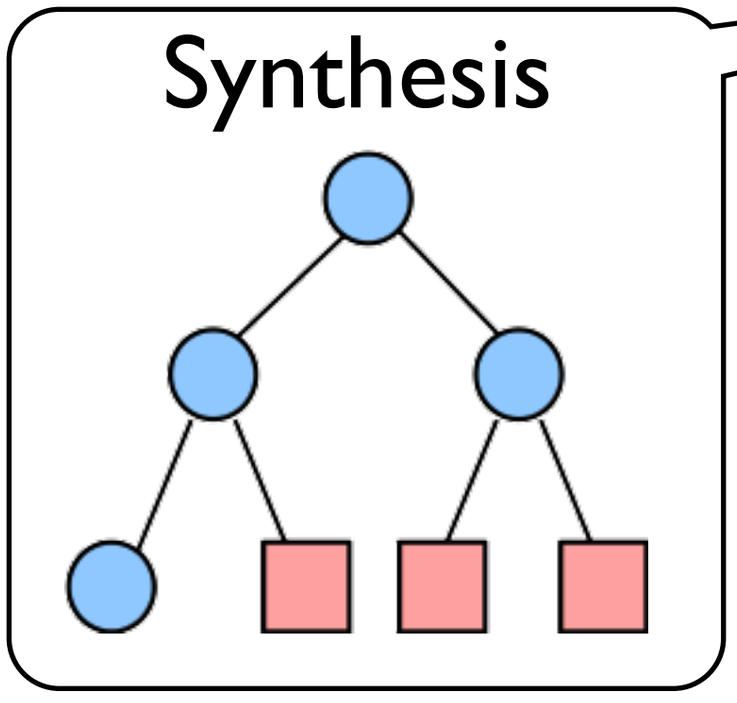
Repair



Specification:

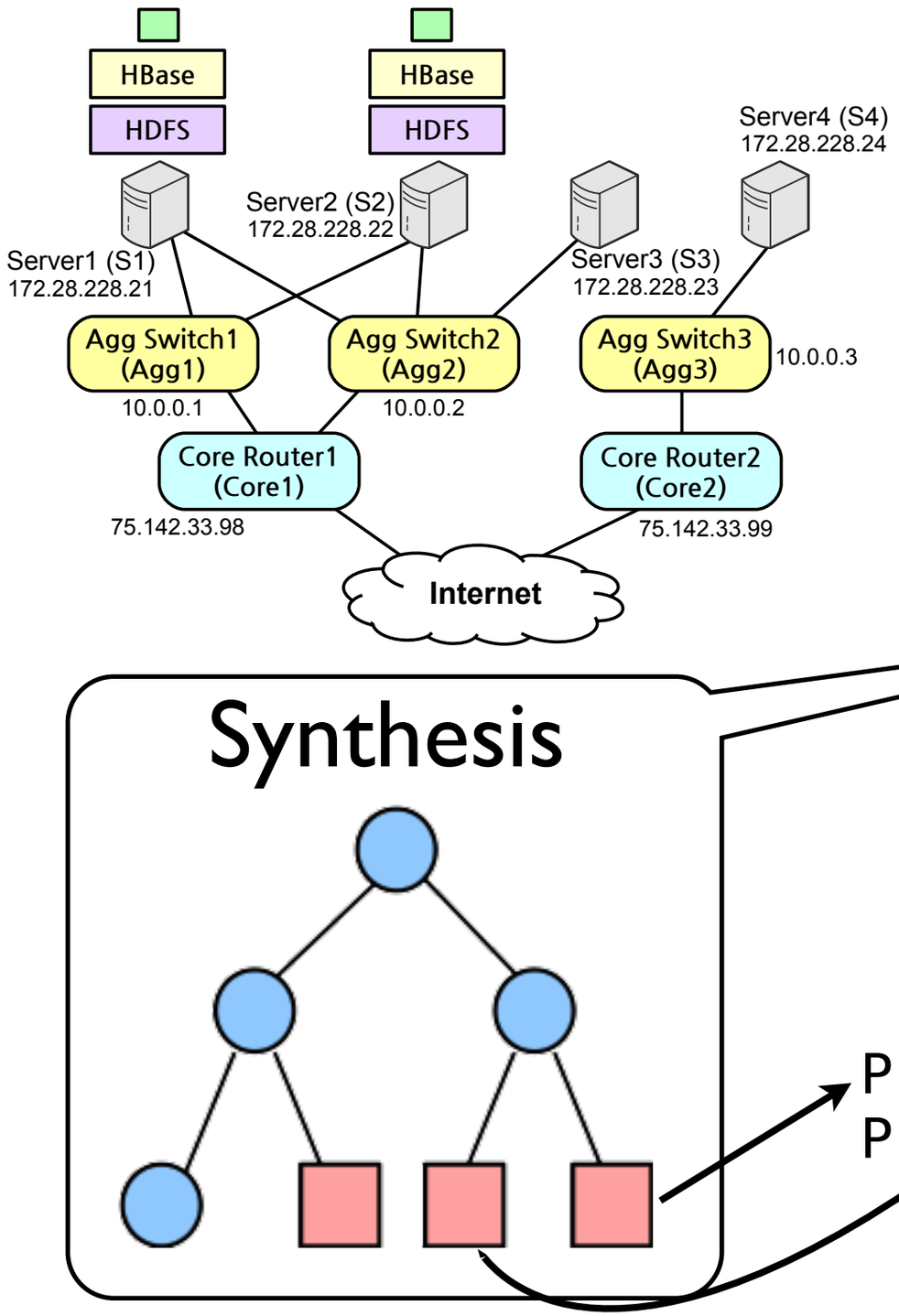
```
$Server -> 172.28.228.21, 172.28.228.22
goal(failProb(ft)<0.08 | ChNode | Agg3)
```






```
Plan 1: Move replica from S1 -> S4
Plan 2: Move replica from S2 -> S4
```

Repair



Specification:

```
$Server -> 172.28.228.21, 172.28.228.22
goal(failProb(ft)<0.08 | ChNode | Agg3)
```

Repair Engine

```
Plan 1: Move replica from S1 -> S4
Plan 2: Move replica from S2 -> S4
```

Evaluation

- Realistic case studies.
- Evaluating expressiveness of our language
- Comparing fault graph analysis algorithms
- Evaluating efficiency of repair engine
- ... ...

Evaluation

- Realistic case studies.
- Evaluating expressiveness of our language
- Comparing fault graph analysis algorithms
- Evaluating efficiency of repair engine
- ... ...

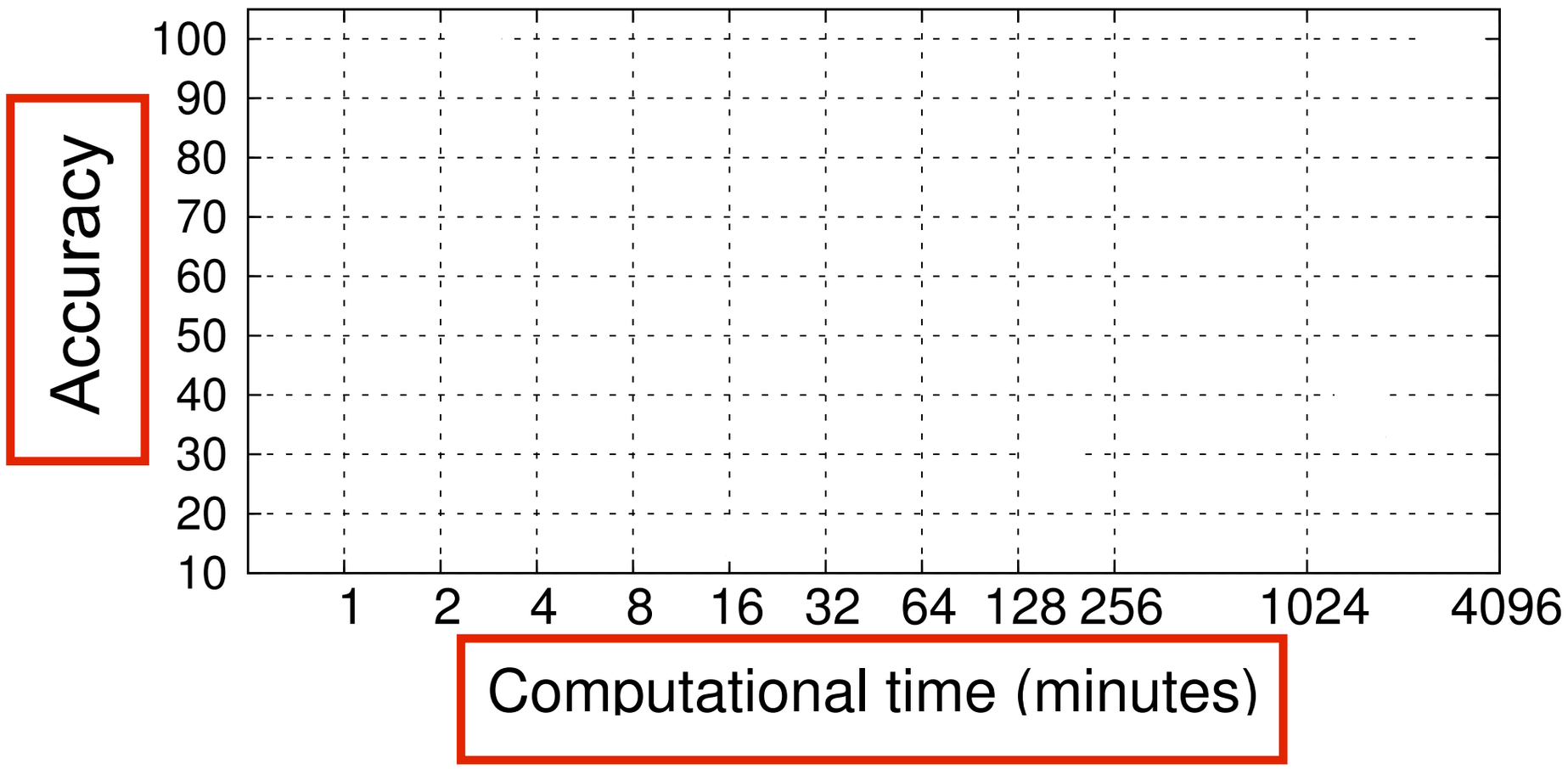
Fault Graph Analysis

	Topology A	Topology B	Topology C
# of Core Routers	144	576	1,024
# of Agg Switches	288	1,152	2,048
# of ToR Switches	288	1,152	2,048
# of Servers	3,456	27,648	65,536
Total # of devices	4,176	30,528	70,656

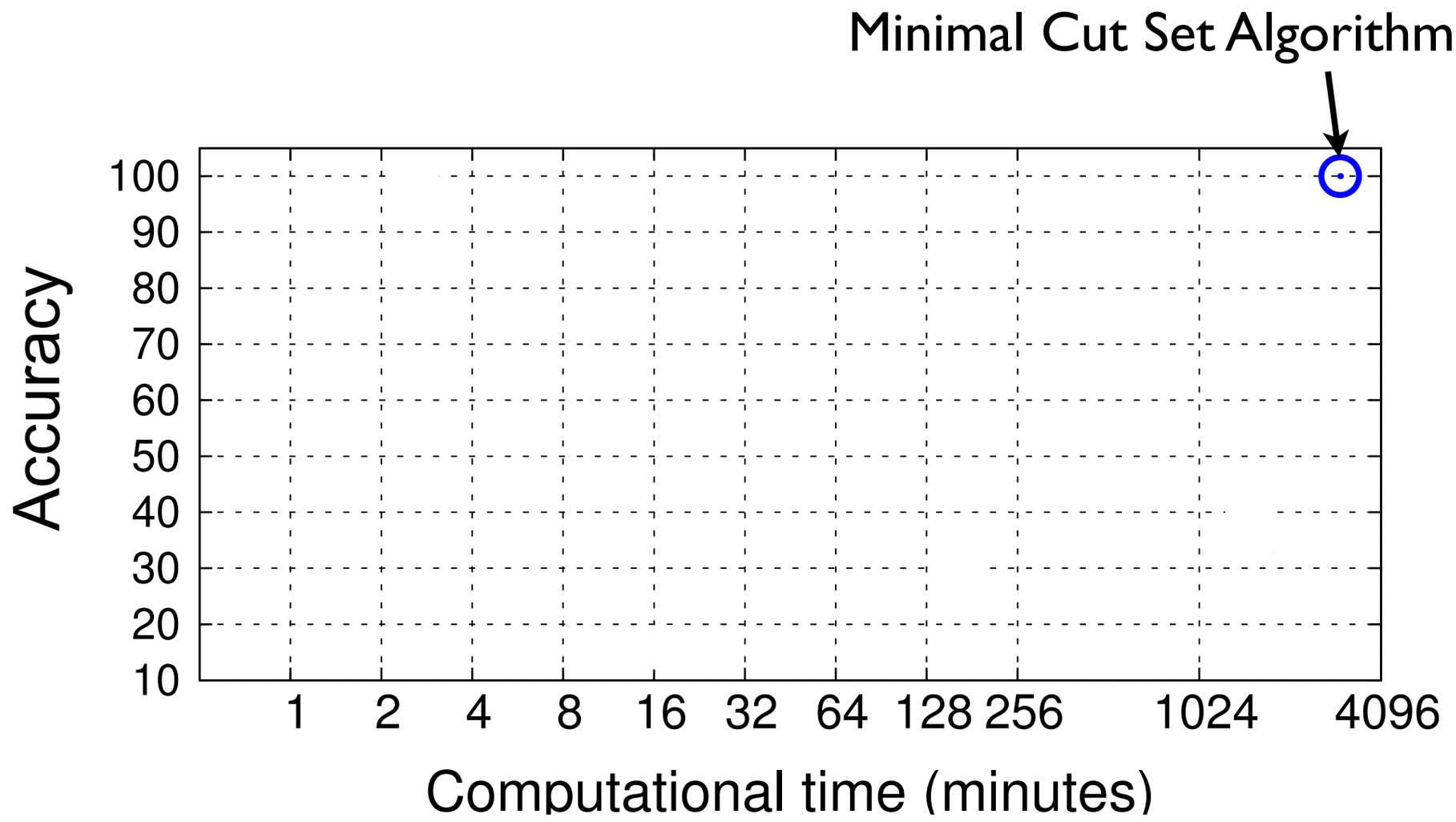
Fault Graph Analysis

	Topology A	Topology B	Topology C
# of Core Routers	144	576	1,024
# of Agg Switches	288	1,152	2,048
# of ToR Switches	288	1,152	2,048
# of Servers	3,456	27,648	65,536
Total # of devices	4,176	30,528	70,656

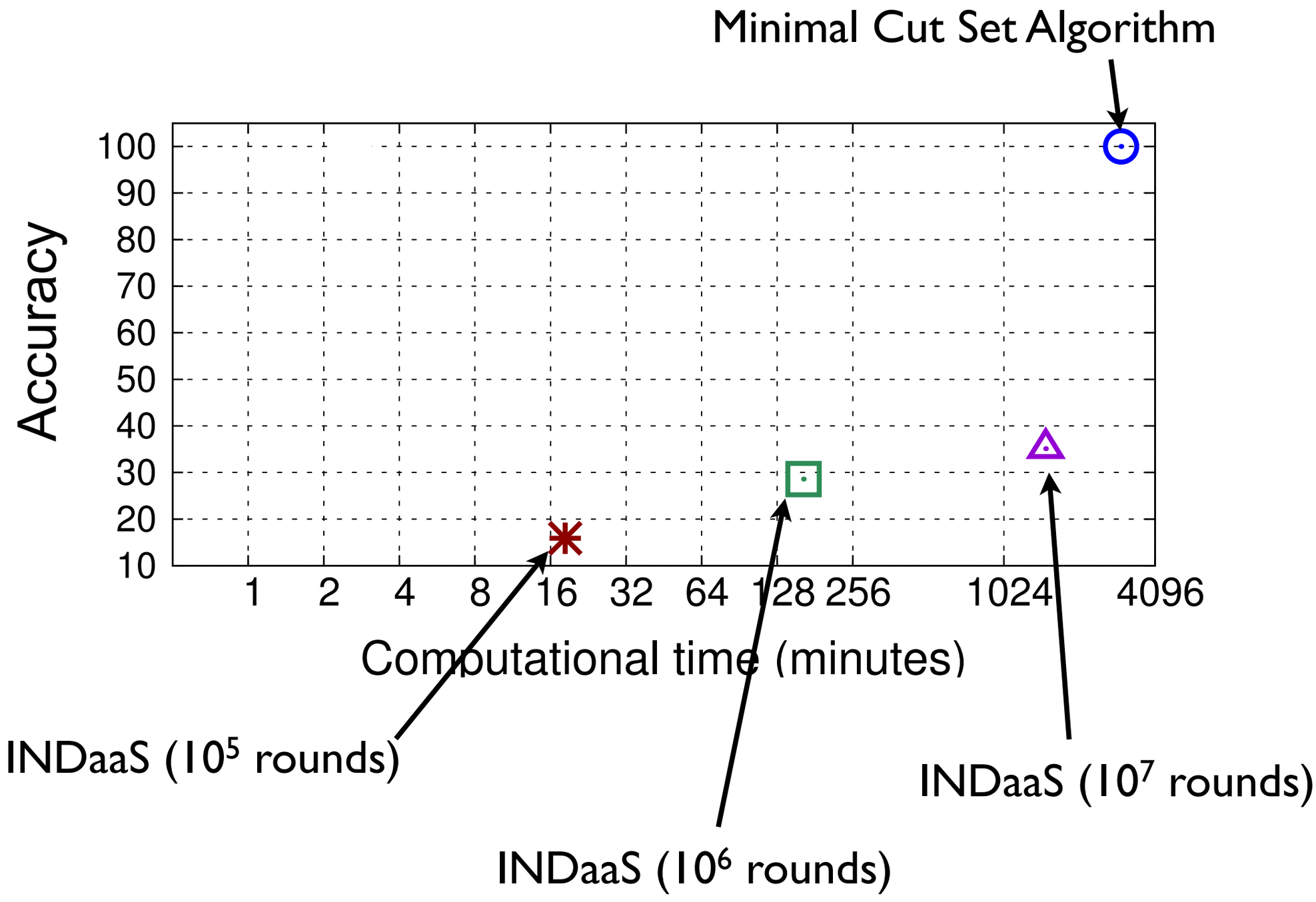
Topology C: 70,656 Nodes



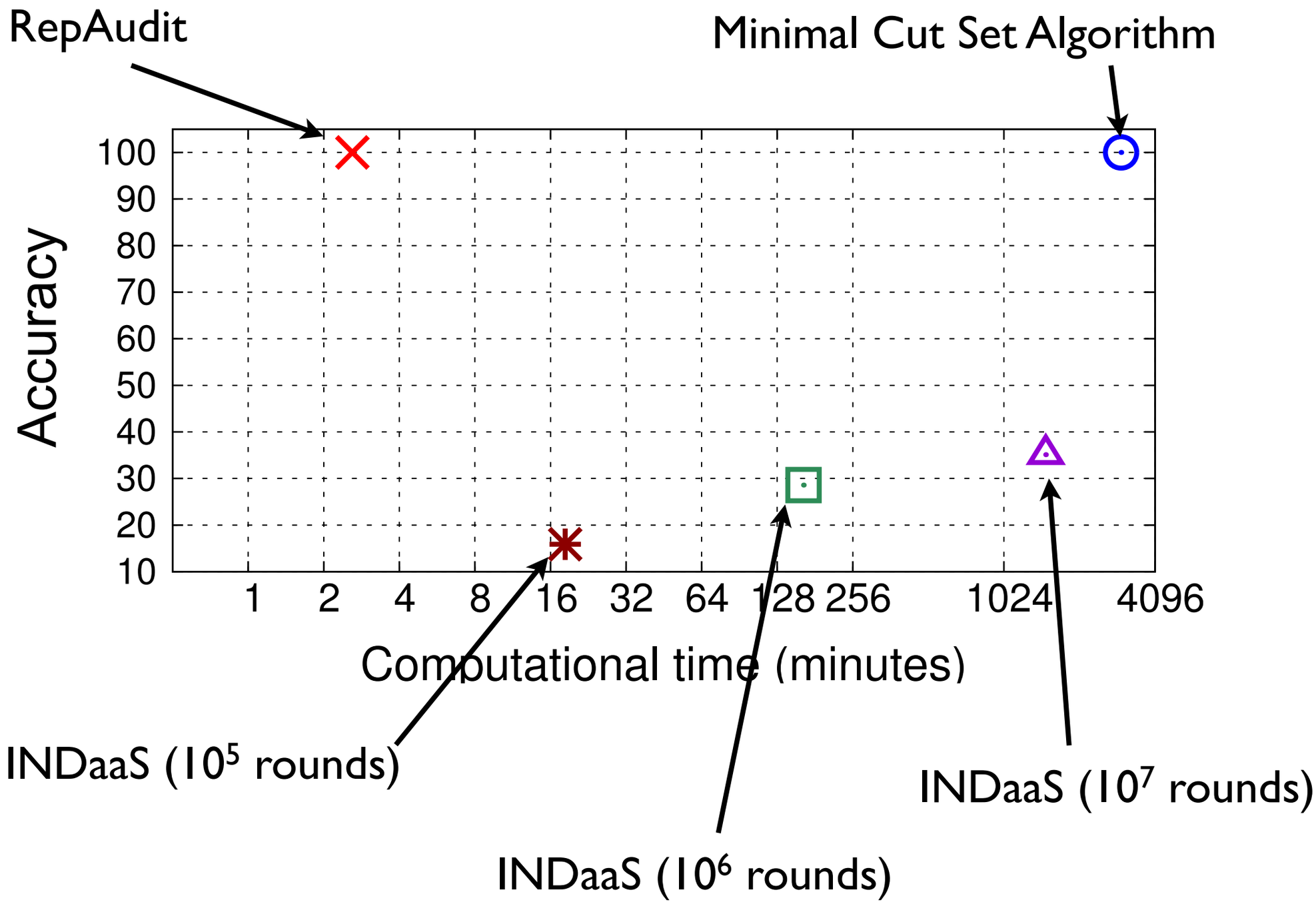
Topology C: 70,656 Nodes



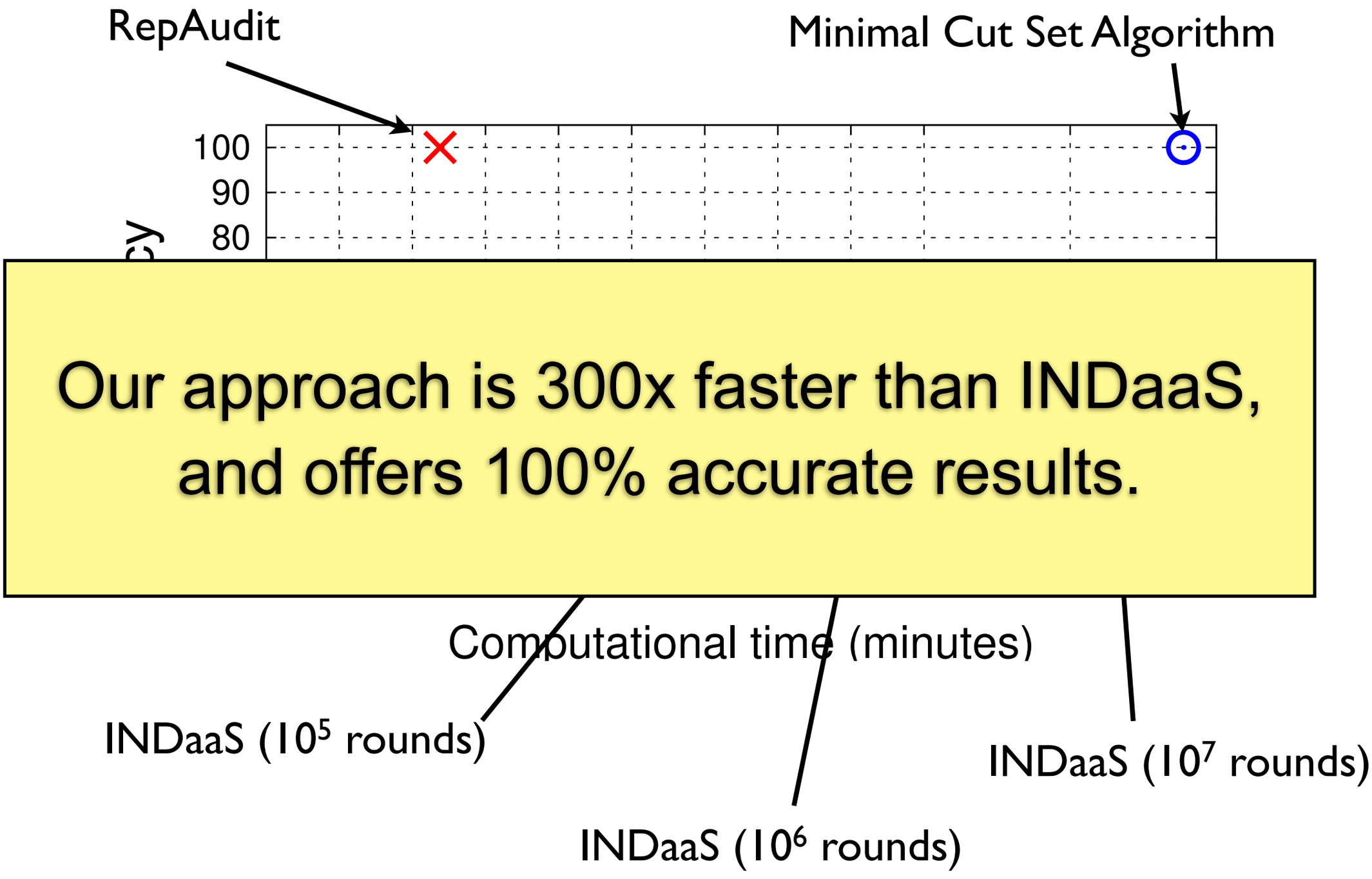
Topology C: 70,656 Nodes



Topology C: 70,656 Nodes



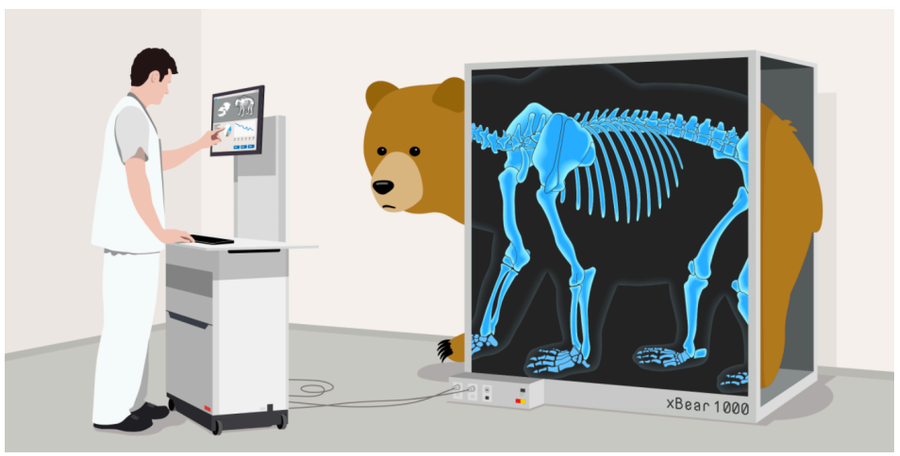
Topology C: 70,656 Nodes



Conclusion

- INDaaS is the first system preventing cascading failures
  - Automatically collecting dependency data
  - Reasonable abstraction: Fault graph

- RepAudit is a language framework auditing cascading failures in system runtime:
  - Flexible to express diverse auditing tasks
  - Accurate and rapid auditing capabilities
  - Useful to build new applications (e.g., repair)



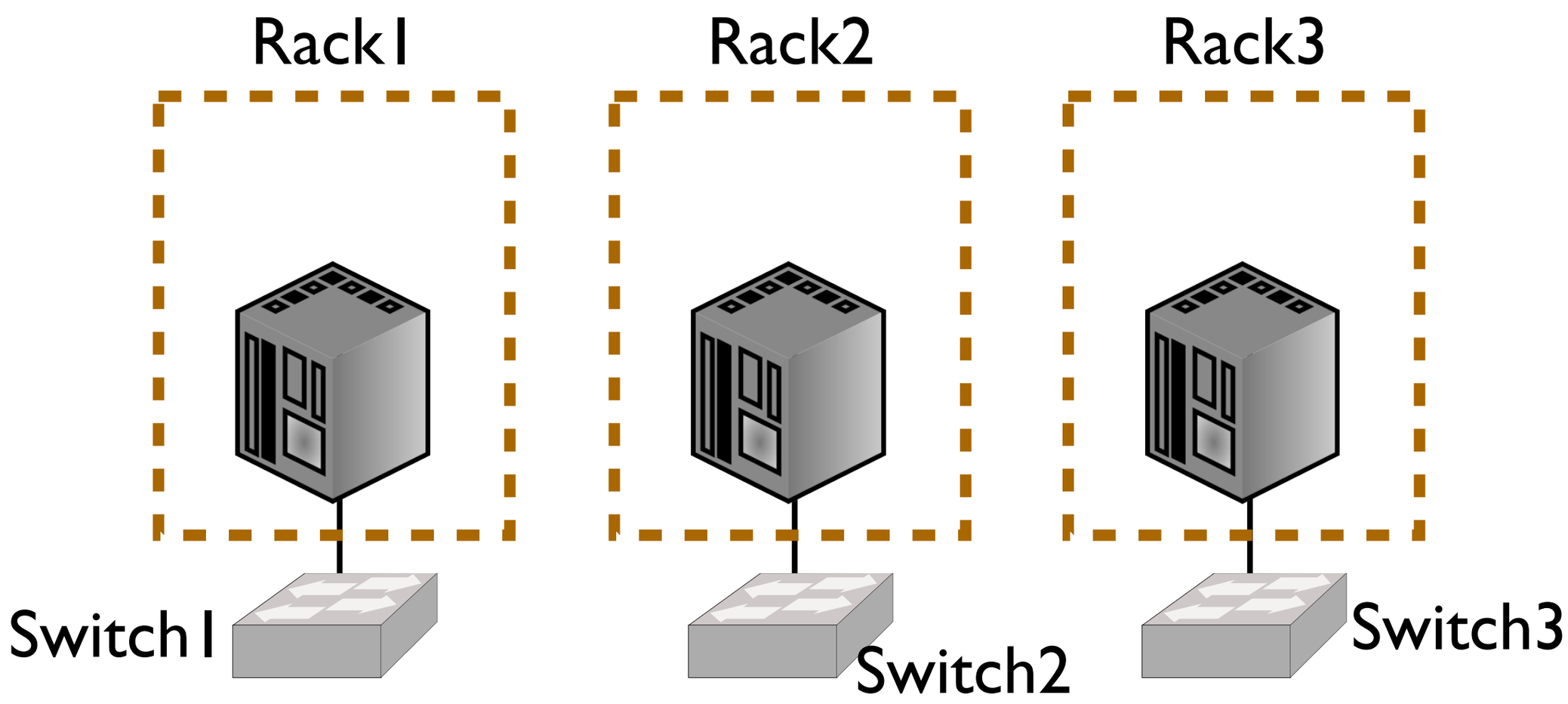
Thanks, questions?

- INDaaS + RepAudit: Preventing cascading failures

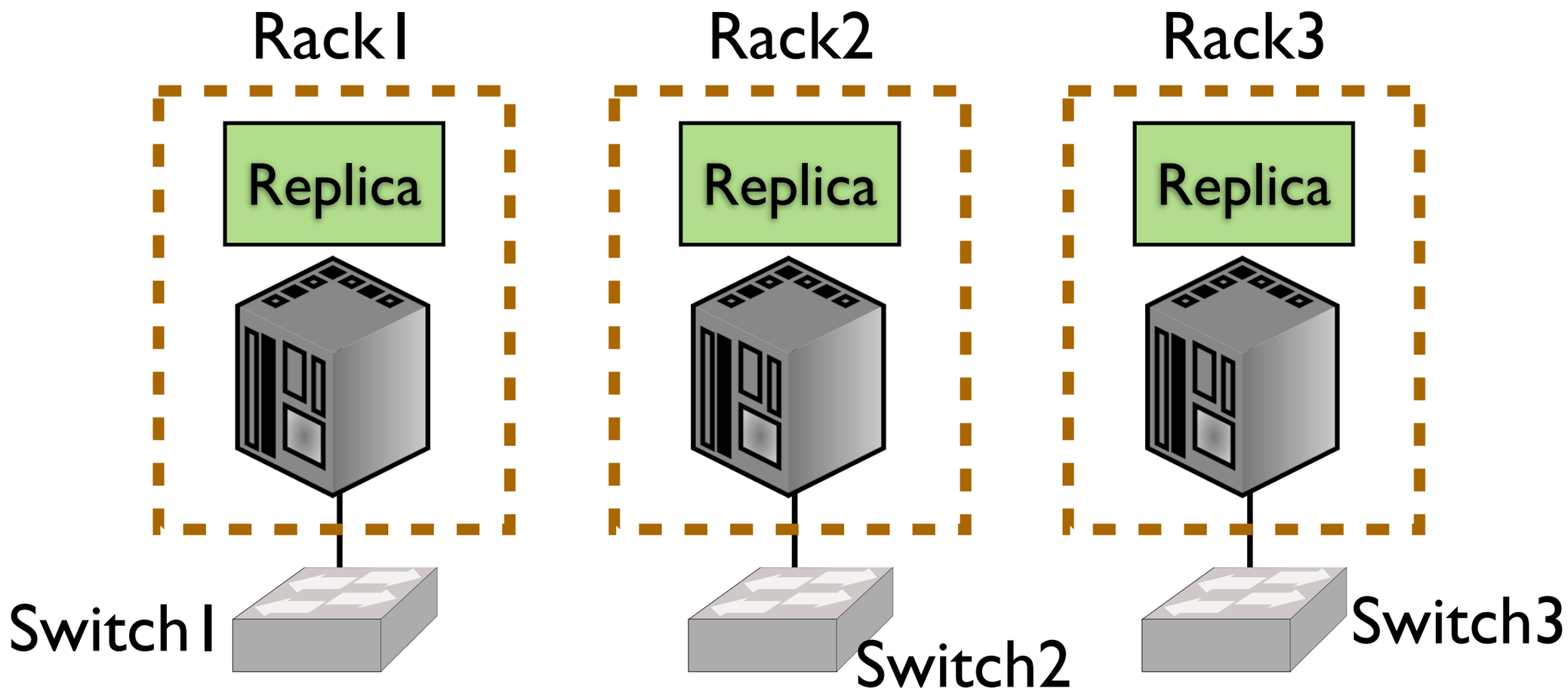
- Find source code at:
  - http://github.com/ennanzhai/repaudit

- I will be at the poster session

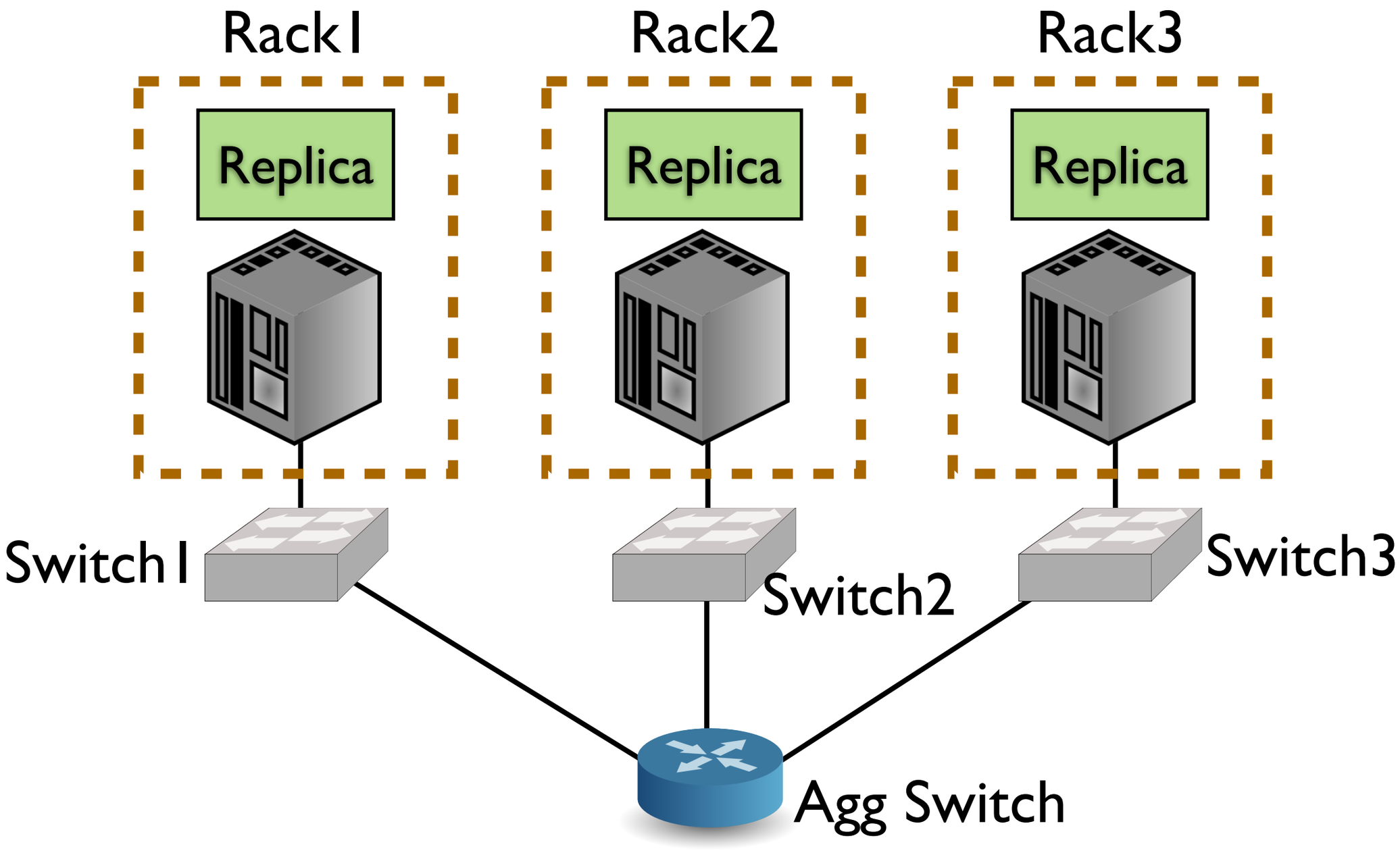
Cascading Failure Example



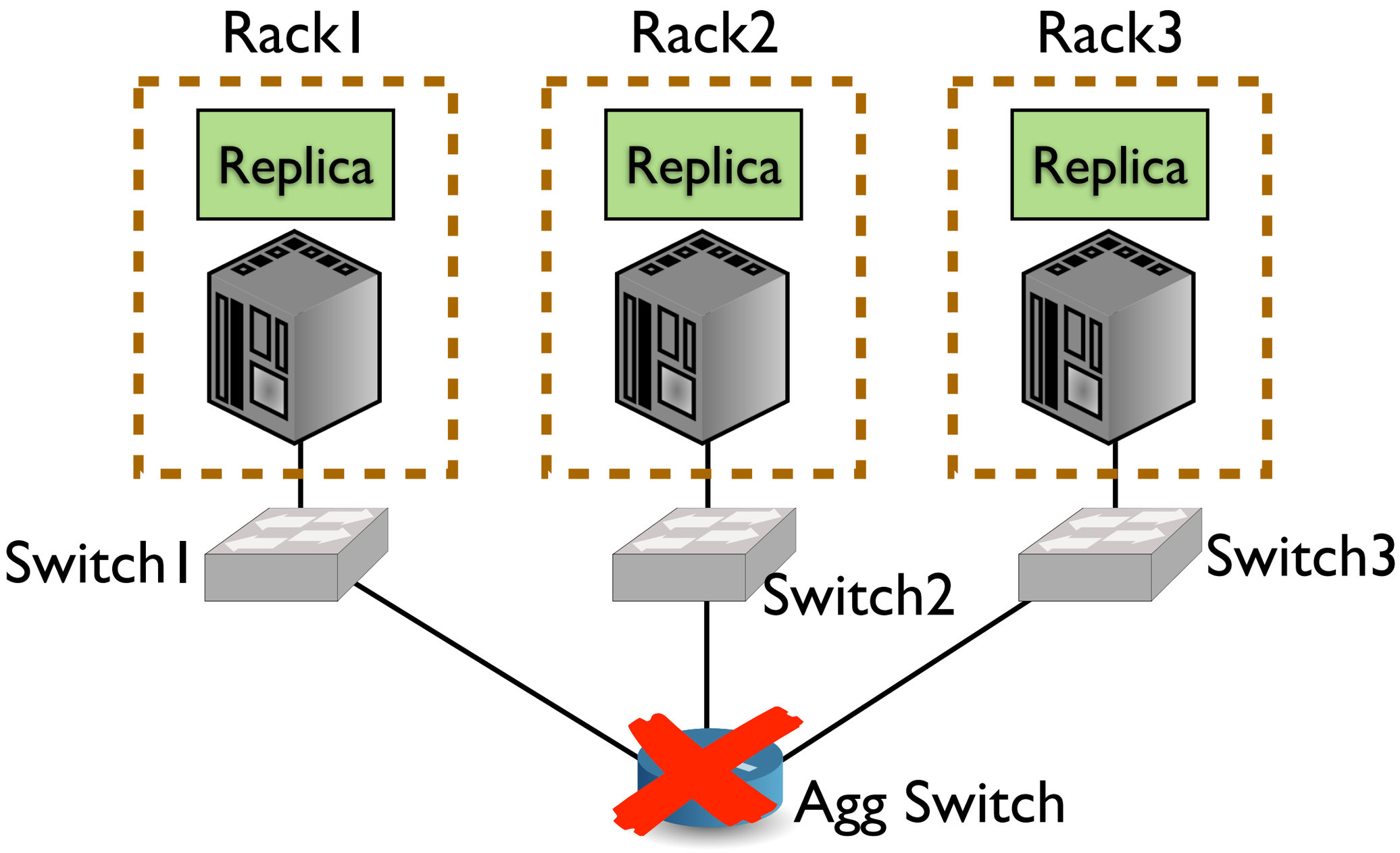
Cascading Failure Example



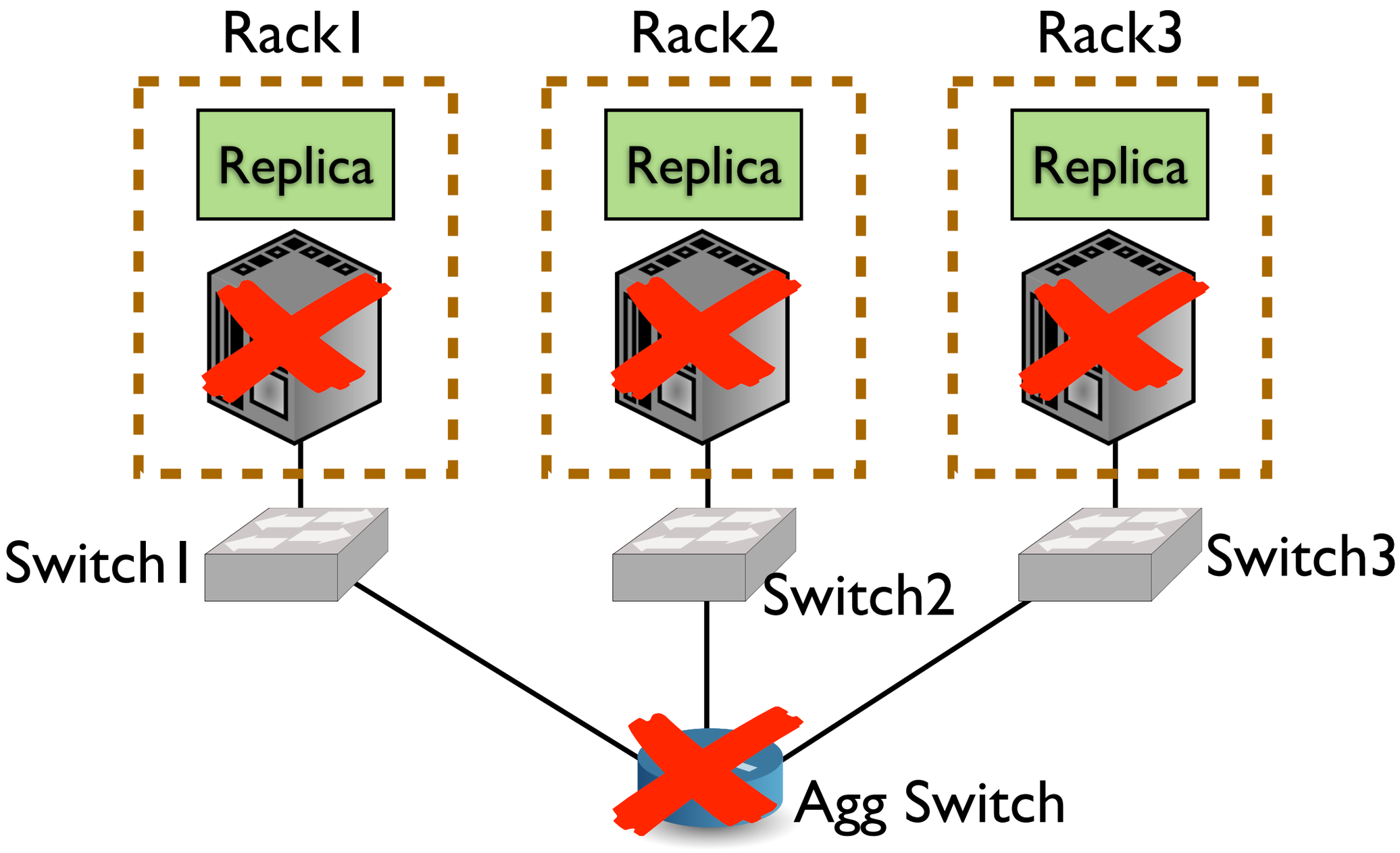
Cascading Failure Example



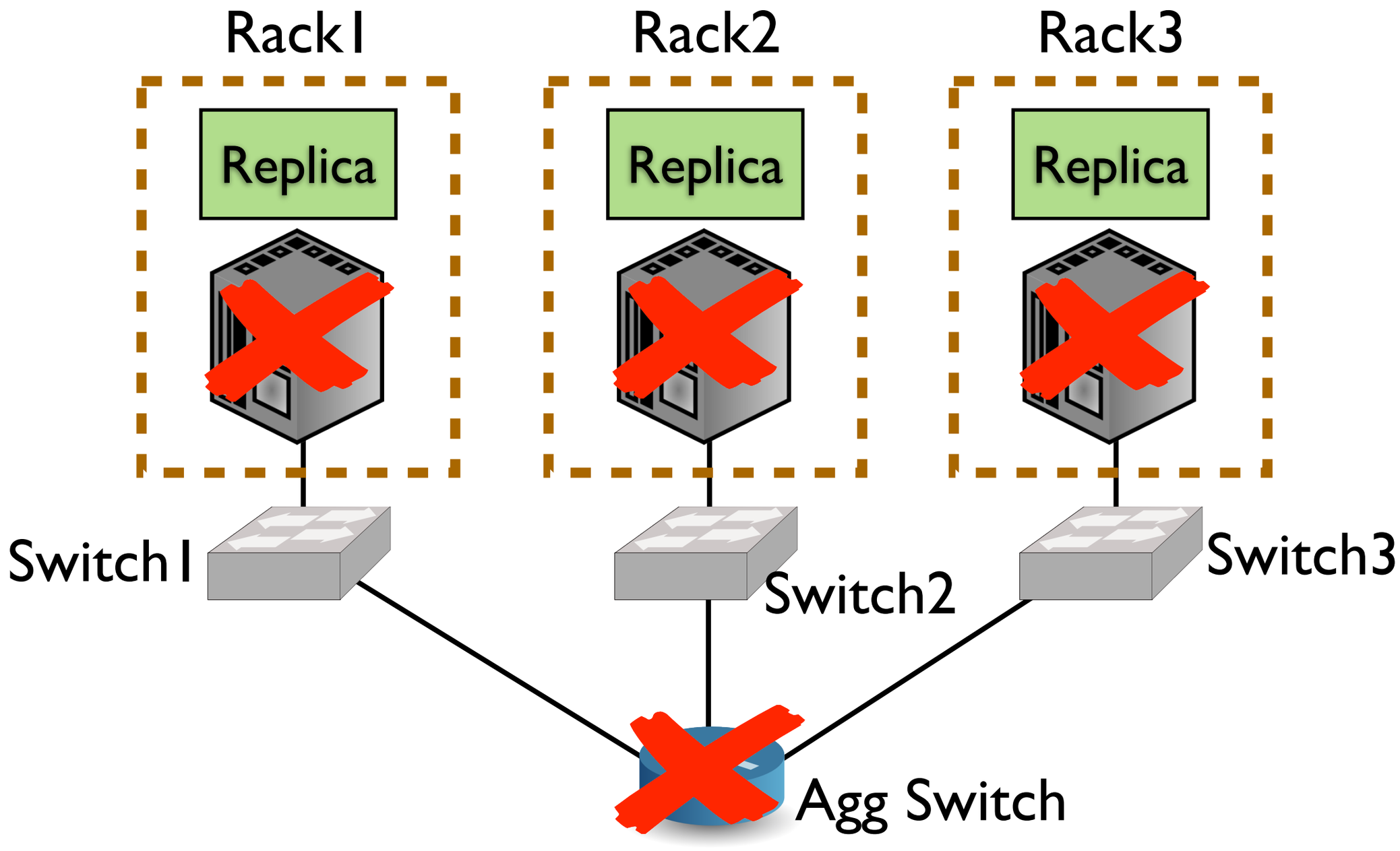
Cascading Failure Example



Cascading Failure Example



Cascading Failure



Cascading Failures in Reality

Cascading Failures in Reality

- Cascading failures by network root causes
- Cascading failures by software root causes

Cascading Failures in Reality

- Cascading failures by network root causes
- Cascading failures by software root causes

Cascading Failures by Network Errors

Rackspace Outage Nov 12th

2 years ago 1,120 Views

On November 12th at 13:51 CST Rackspace experienced an isolated issue in their core network. A small number of their customers were affected, including REW. The outage lasted about 90 minutes. In simple terms, a core network switch died and when the traffic failed over to the secondary switch it also died. Rackspace is investigating the incident to find ways to improve their network and processes to ensure this event is not repeated. REW Sysadmins were immediately notified of the outage by our monitoring tools and were in constant contact with Rackspace during the outage working to resolve as quickly as possible.

REW apologizes for this outage; we promise that we are putting Rackspace's feet to the fire to ensure maximum uptime for our customers!

Here is the incident report from Rackspace if you want the techy details:

http://www.realestatewebmasters.com/blogs/rew-steven/rackspace-outage-nov-12th/show/

Cascading Failures in Reality

- Cascading failures by network root causes
- Cascading failures by software root causes

Discovering Critical Risk Groups

- Discovering the top-k risk groups with the highest failure probabilities
  - We want to maximize $C = \prod c_i \cdot w_i$ rather than $C = \sum c_i \cdot w_i$
  - Use *(-100)log* c_i as the cost

{A1=0.2, A2=0.6} and {A3=0.39, A3=0.39}

log0.2+log0.6 = log 0.2*0.6

log0.39+log0.39 = log 0.39*0.39

Discovering Critical Risk Groups

- Discovering the top-k risk groups with the highest failure probabilities
  - We want to maximize $C = \prod c_i \cdot w_i$ rather

Our engine is 300x faster than INDaaS, and offers 100% accurate results.

log0.2+log0.6 = log 0.2*0.6

log0.39+log0.39 = log 0.39*0.39

First-Step System: INDaaS [OSDI'14]