Using SAT Solvers to Prevent Causal Failures in the Cloud

SHONAN SEMINAR 139: CAUSAL REASONING IN SYSTEMS

RUZICA PISKAC

YALE UNIVERSITY

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

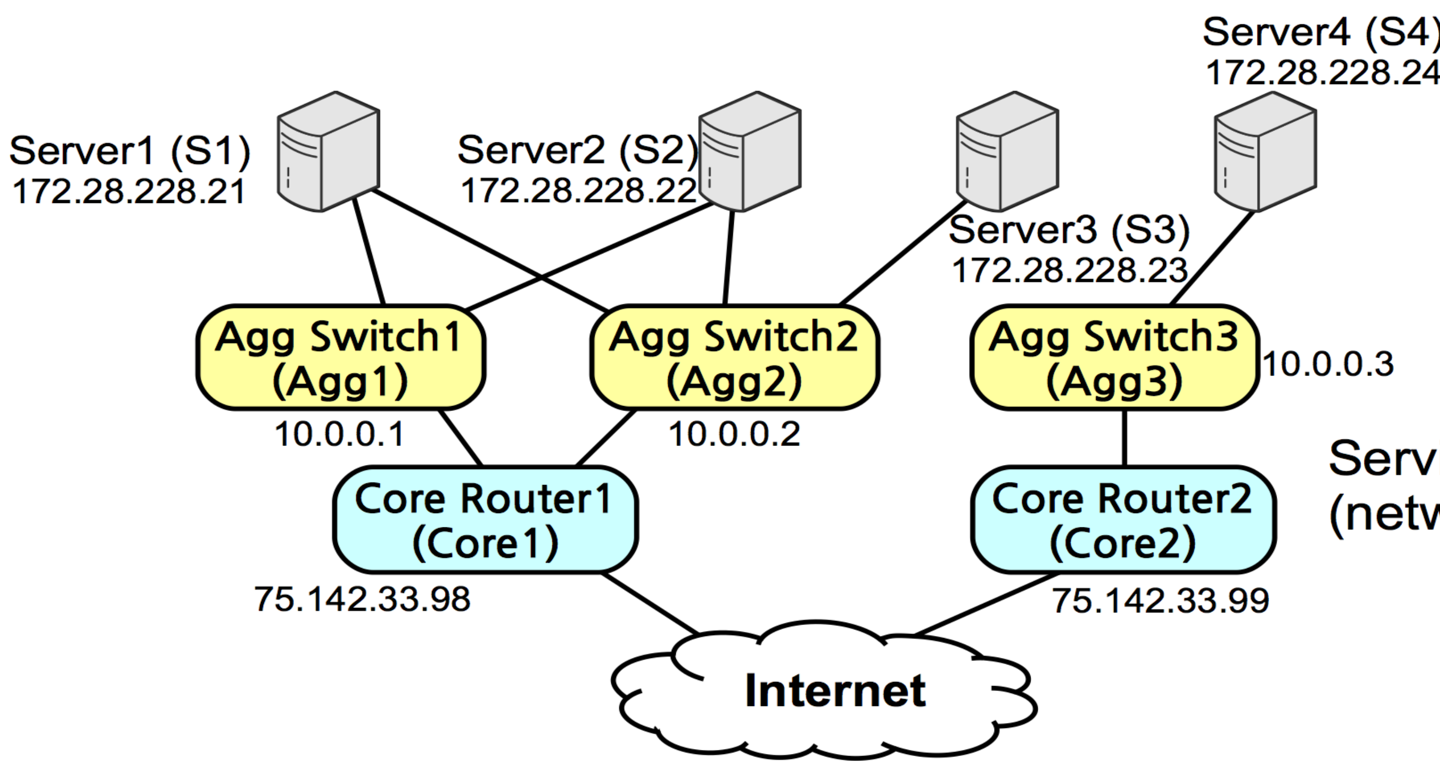
Background

- 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

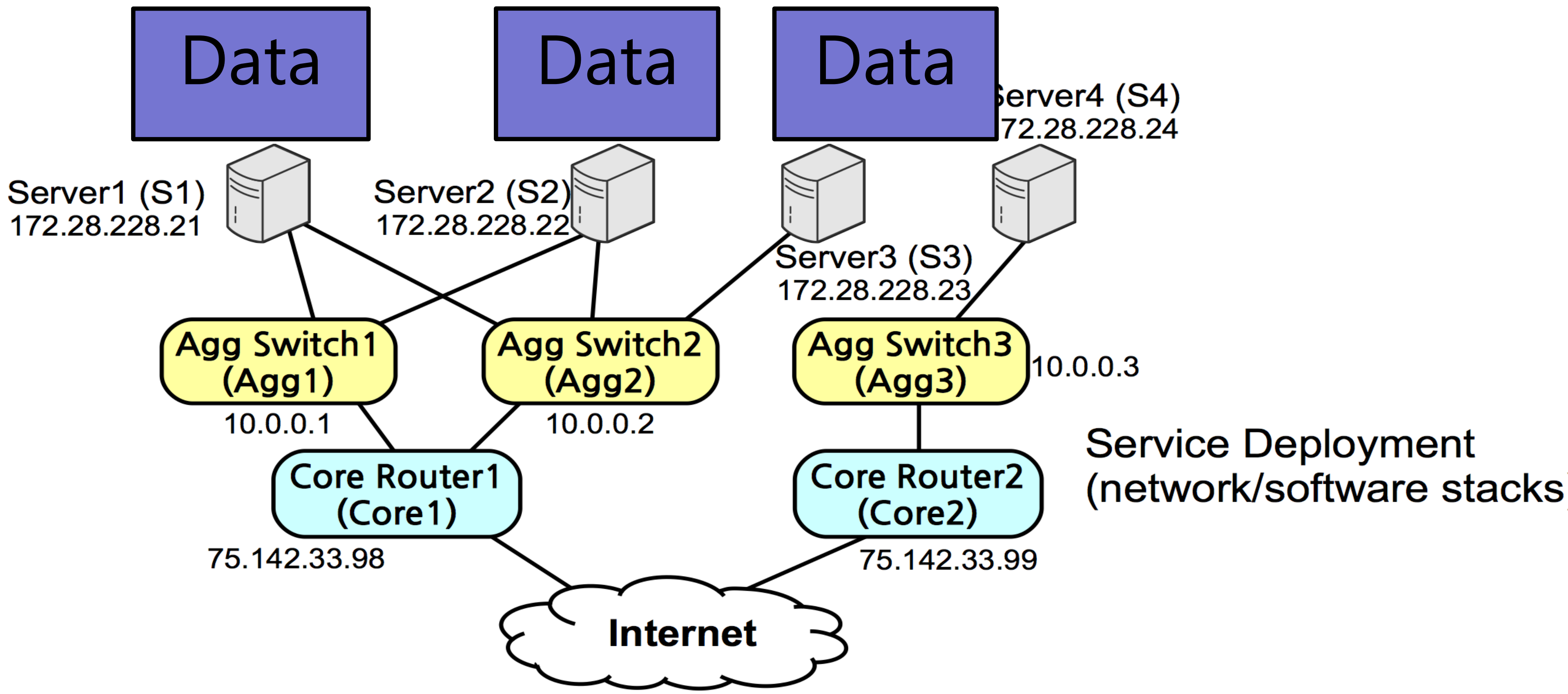
Can replication systems indeed help in obtaining reliability?

Background

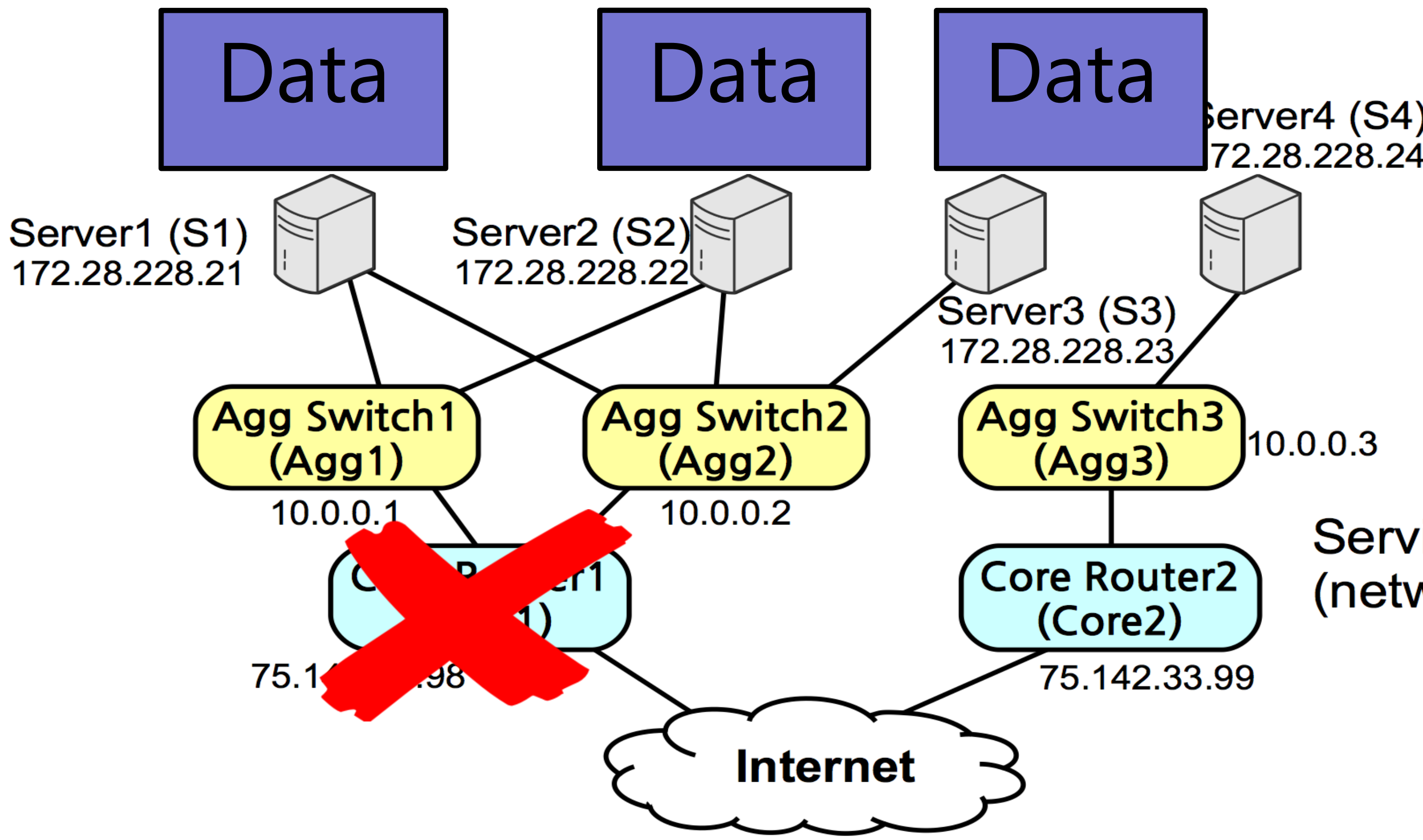


Service Deployment
(network/software stacks)

Background

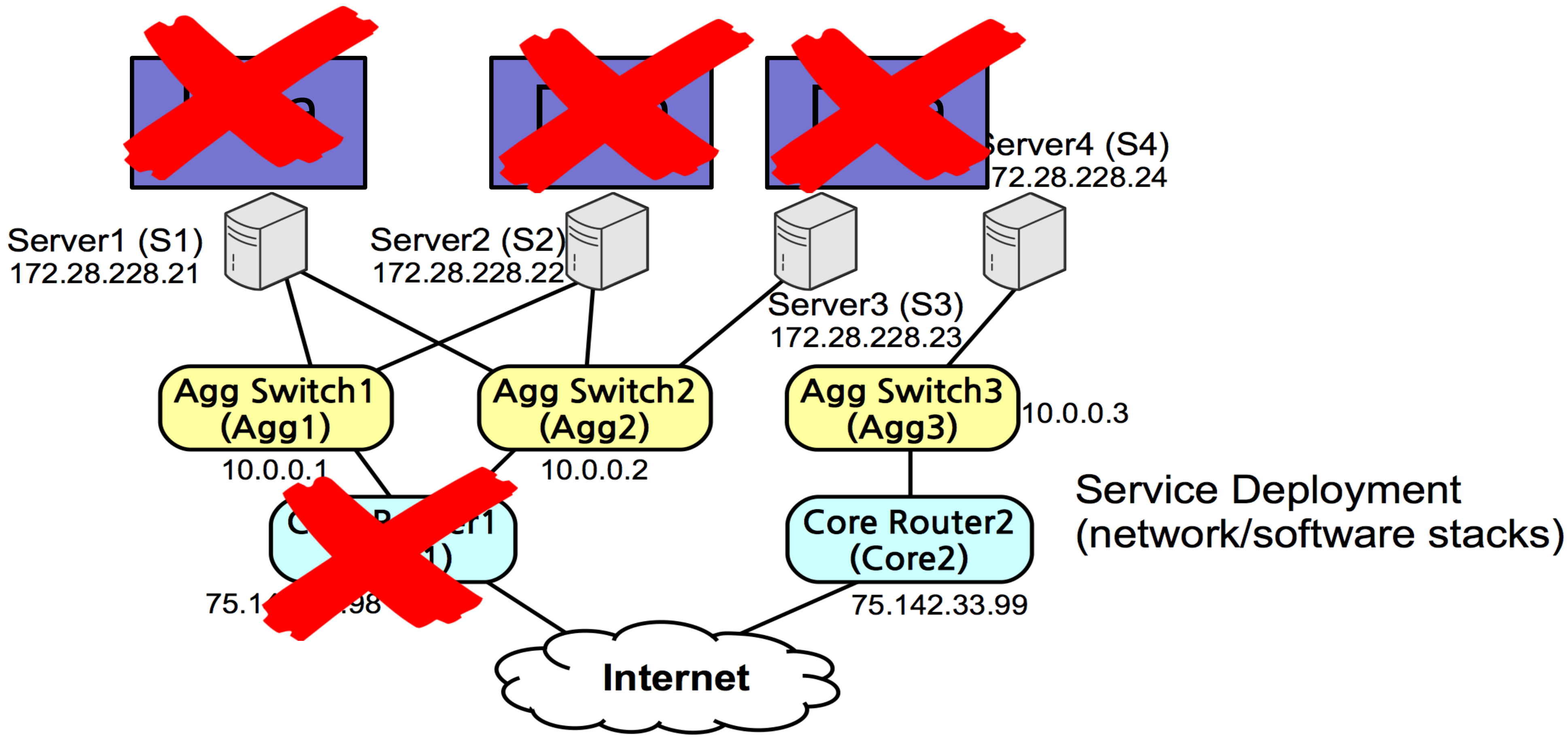

Data
Data
Data
Server1 (S1)
172.28.228.21
Server2 (S2)
172.28.228.22
Server3 (S3)
172.28.228.23
erver4 (S4)
72.28.228.24
Agg Switch1
(Agg1)
10.0.0.1
Agg Switch2
(Agg2)
10.0.0.2
Agg Switch3
(Agg3)
10.0.0.3
Core Router1
(Core1)
75.142.33.98
Core Router2
(Core2)
75.142.33.99
Internet
Service Deployment
(network/software stacks)

Background



Service Deployment
(network/software stacks)

Background


Server1 (S1)
172.28.228.21
Server2 (S2)
172.28.228.22
Server3 (S3)
172.28.228.23
erver4 (S4)
72.28.228.24
Agg Switch1
(Agg1)
10.0.0.1
Agg Switch2
(Agg2)
10.0.0.2
Agg Switch3
(Agg3)
10.0.0.3
Core Router2
(Core2)
75.142.33.99
Internet
Service Deployment
(network/software stacks)

Real-World Correlated Failures

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)

Real-World Correlated Failures

amazon web services™

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

Summary of t

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

The Primary
and Amazon

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:

Real-World Correlated Failures

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
raffic failed over to the
t 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:

Existing Approaches

- Cloud providers handle correlated 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 correlated 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

Disease prevention is better than diagnosis

-- *World Health Organization*

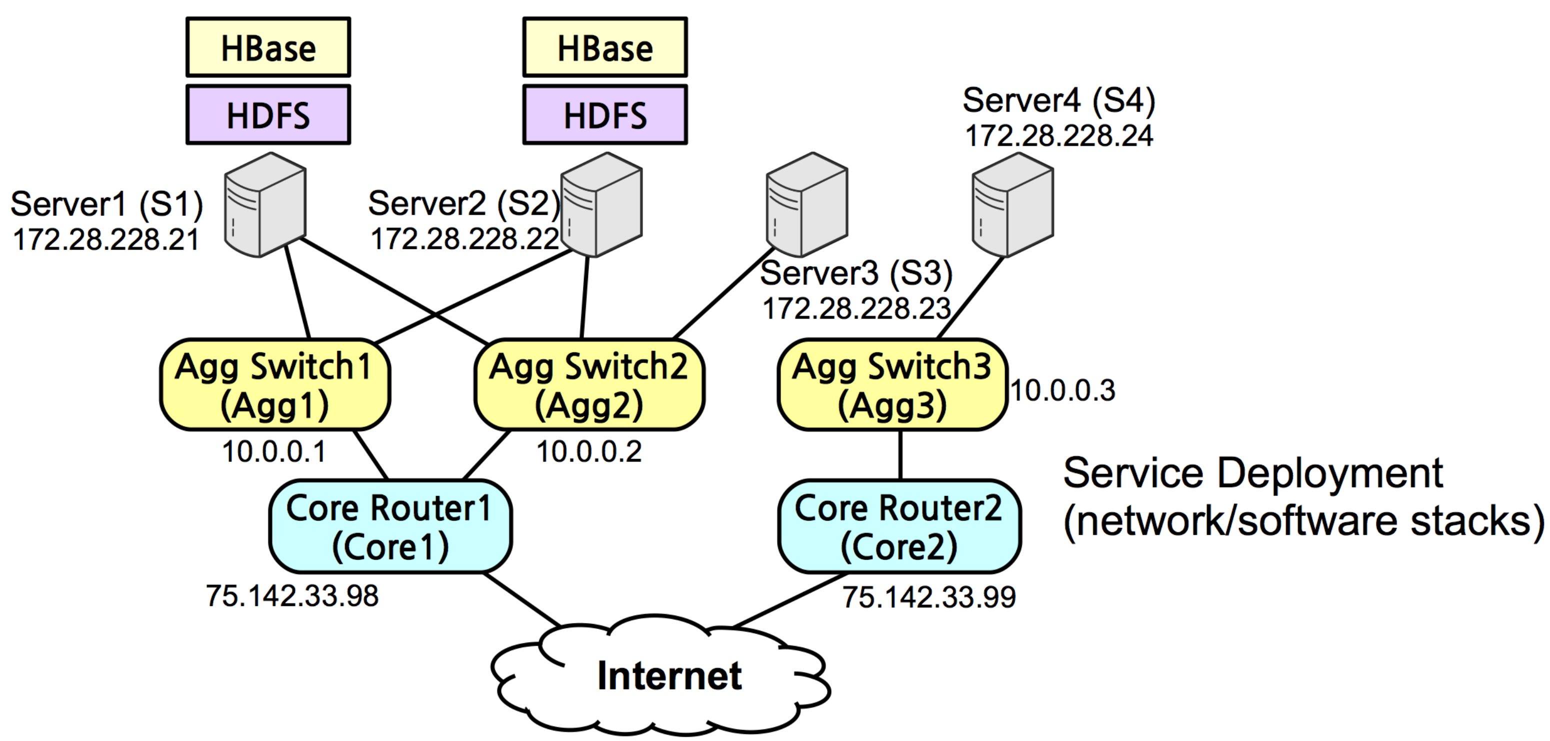
Goal of this Project: Preventing Correlated Failures

- INDaaS: First effort towards this goal [OSDI'14]
  - Heading off correlated failures through Independence-as-a-Service

- This work: an auditing language framework RepAudit
  - An auditing language for preventing correlated failures within the clouds

Initial Motivation: 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



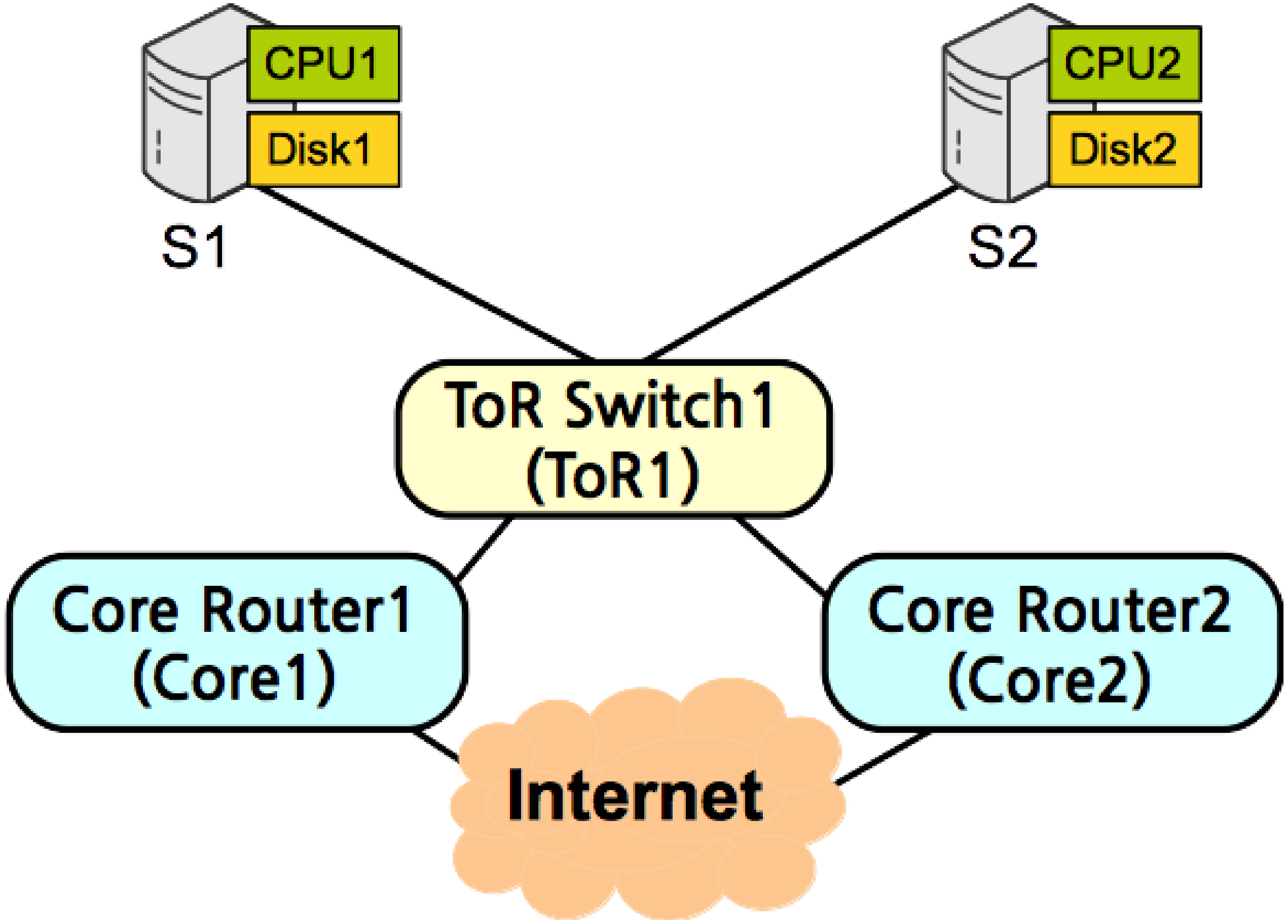
Dependency Data Collections

- Reuse existing data collection tools:
  - Convert the outputs to uniform format.
  - Three types of format: NET, HW and SW.

Our defined format

Type	Dependency Expression
Network	<src=”S” dst=”D” route=”x,y,z”/>
Hardware	<hw=”H” type=”T” dep=”x”/>
Software	<pgm=”S” hw=”H” dep=”x,y,z”/>

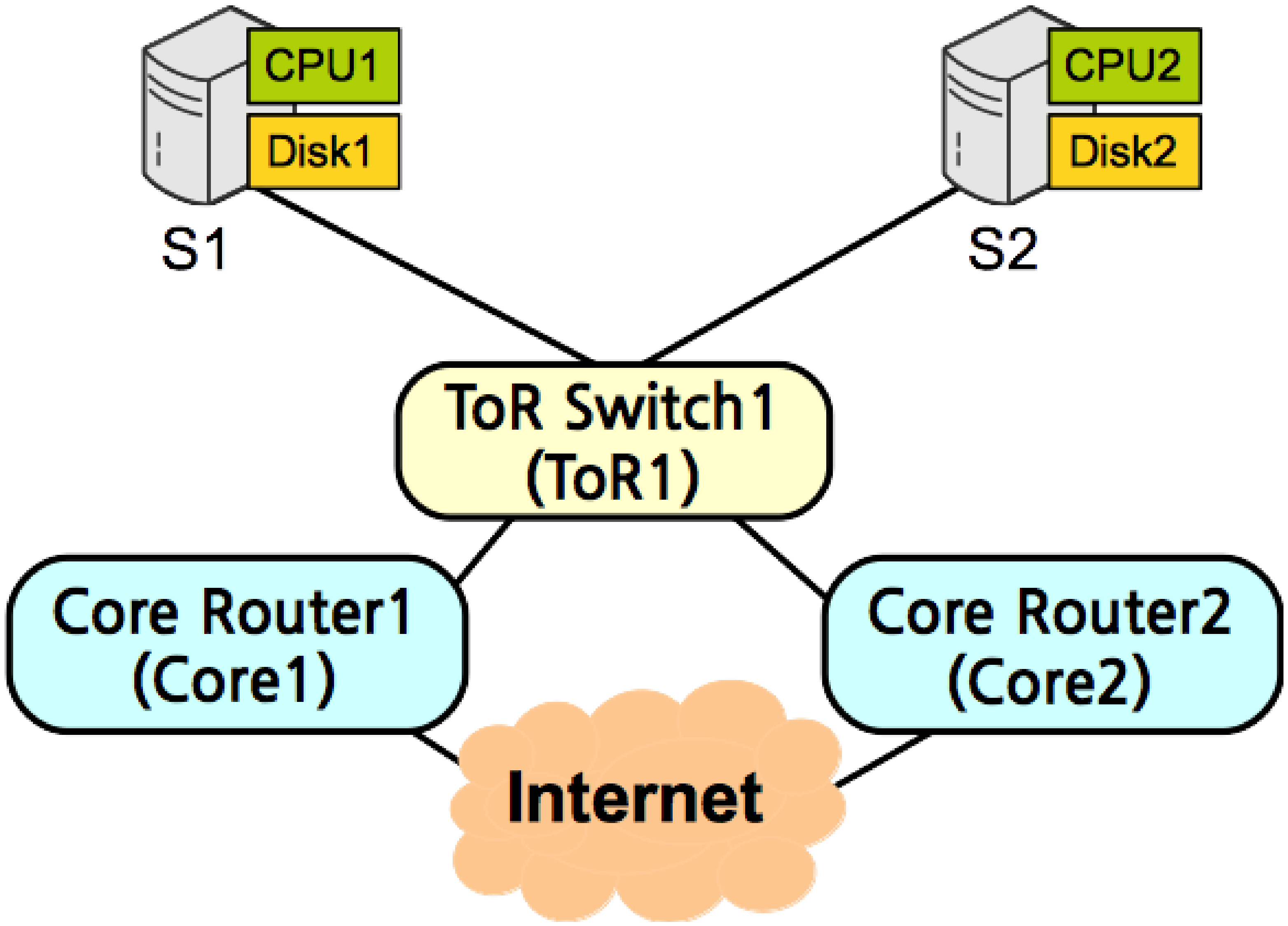
Example



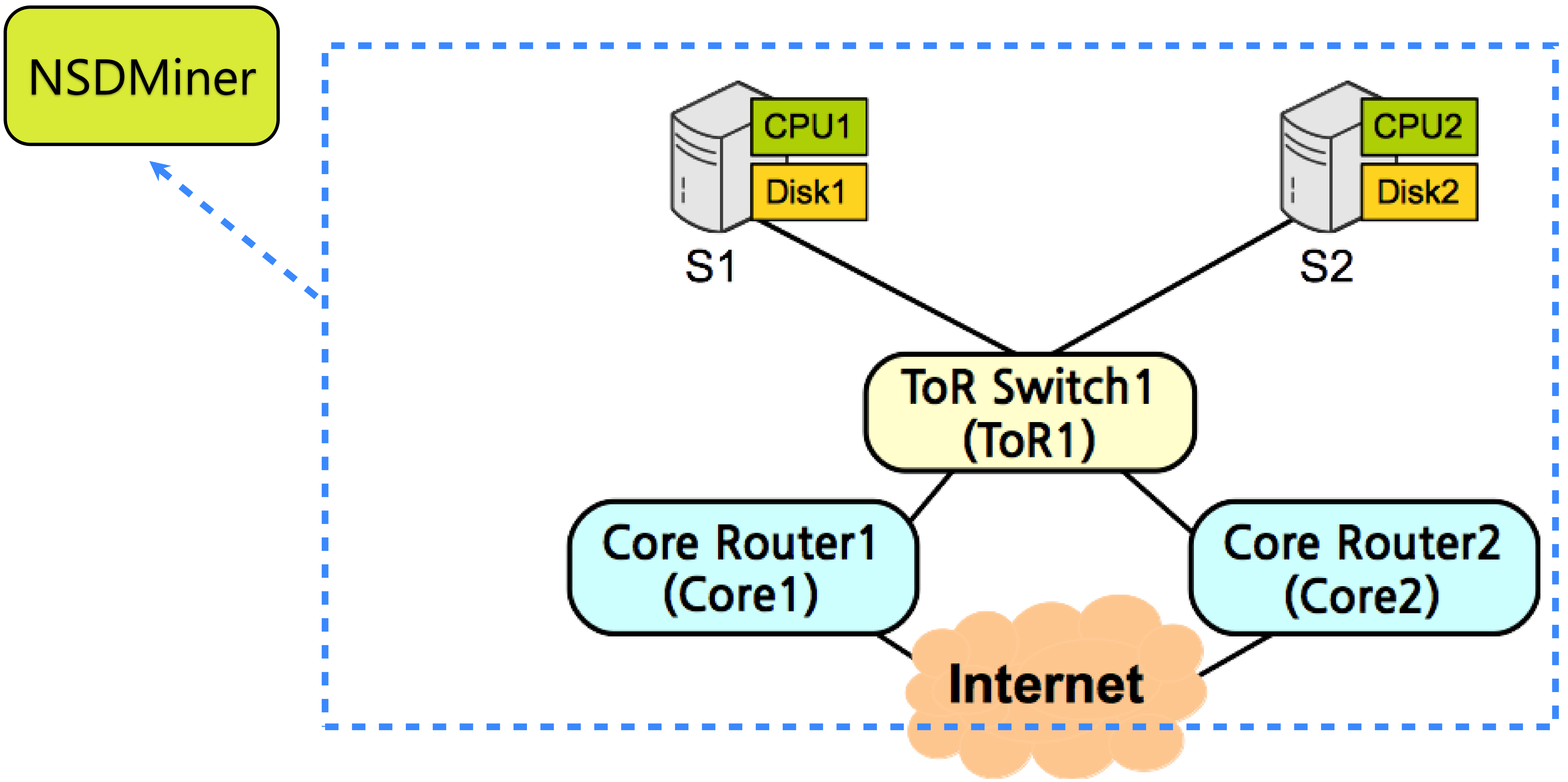
CPU1
Disk1
S1
CPU2
Disk2
S2
ToR Switch1
(ToR1)
Core Router1
(Core1)
Core Router2
(Core2)
Internet

Example

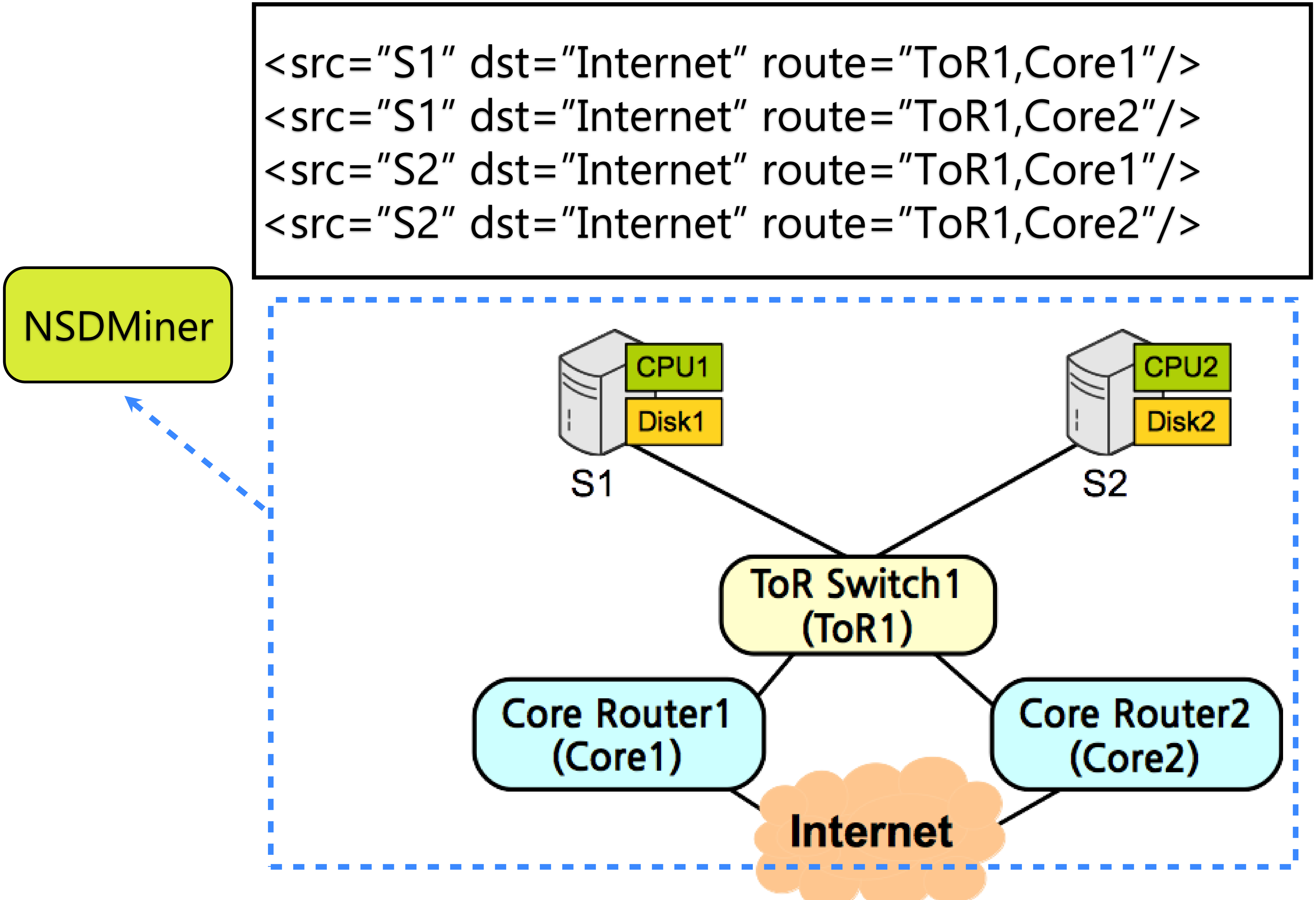
NSDMiner



Example

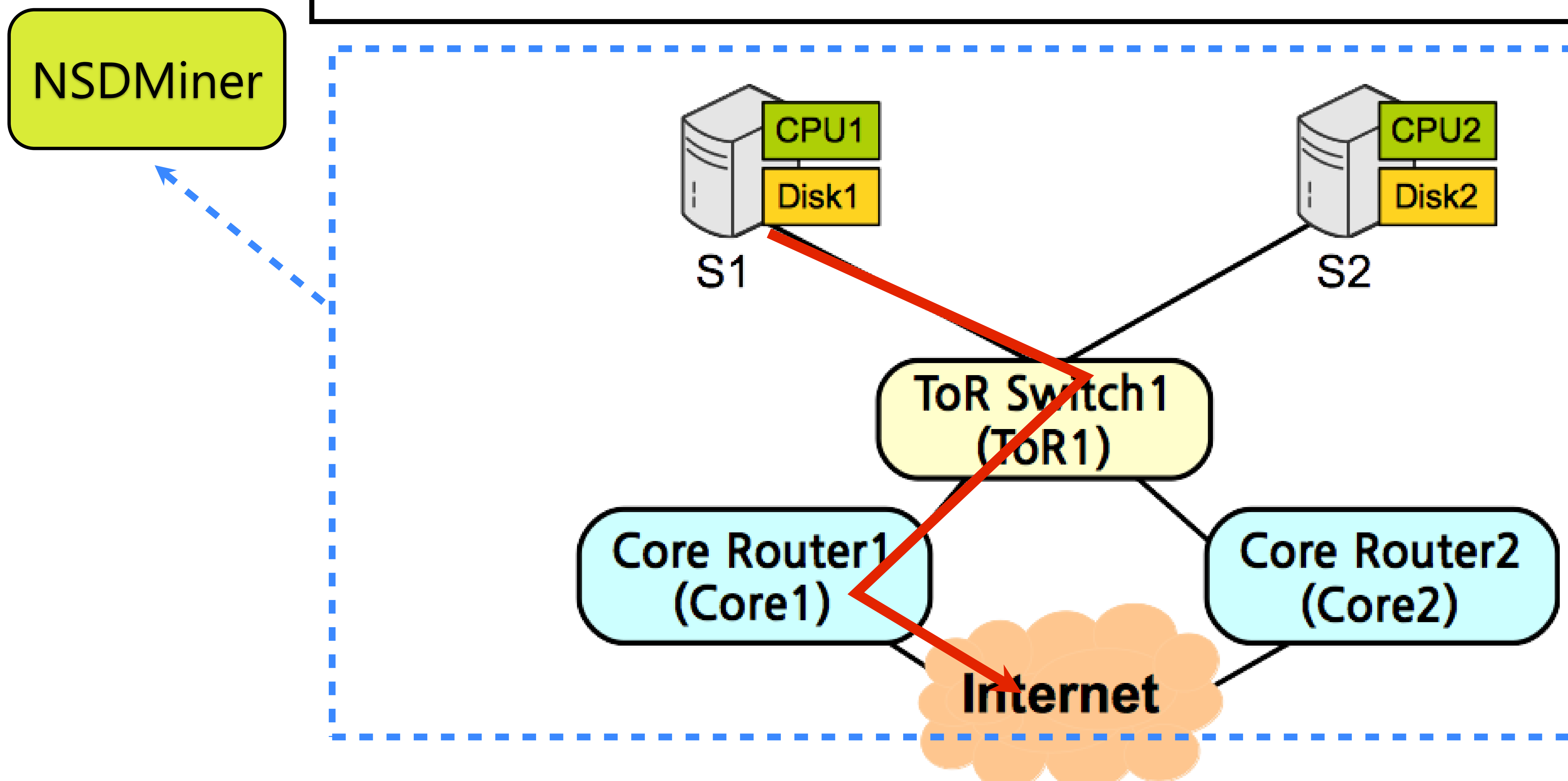


NSDMiner
CPU1
Disk1
S1
CPU2
Disk2
S2
ToR Switch1
(ToR1)
Core Router1
(Core1)
Core Router2
(Core2)
Internet

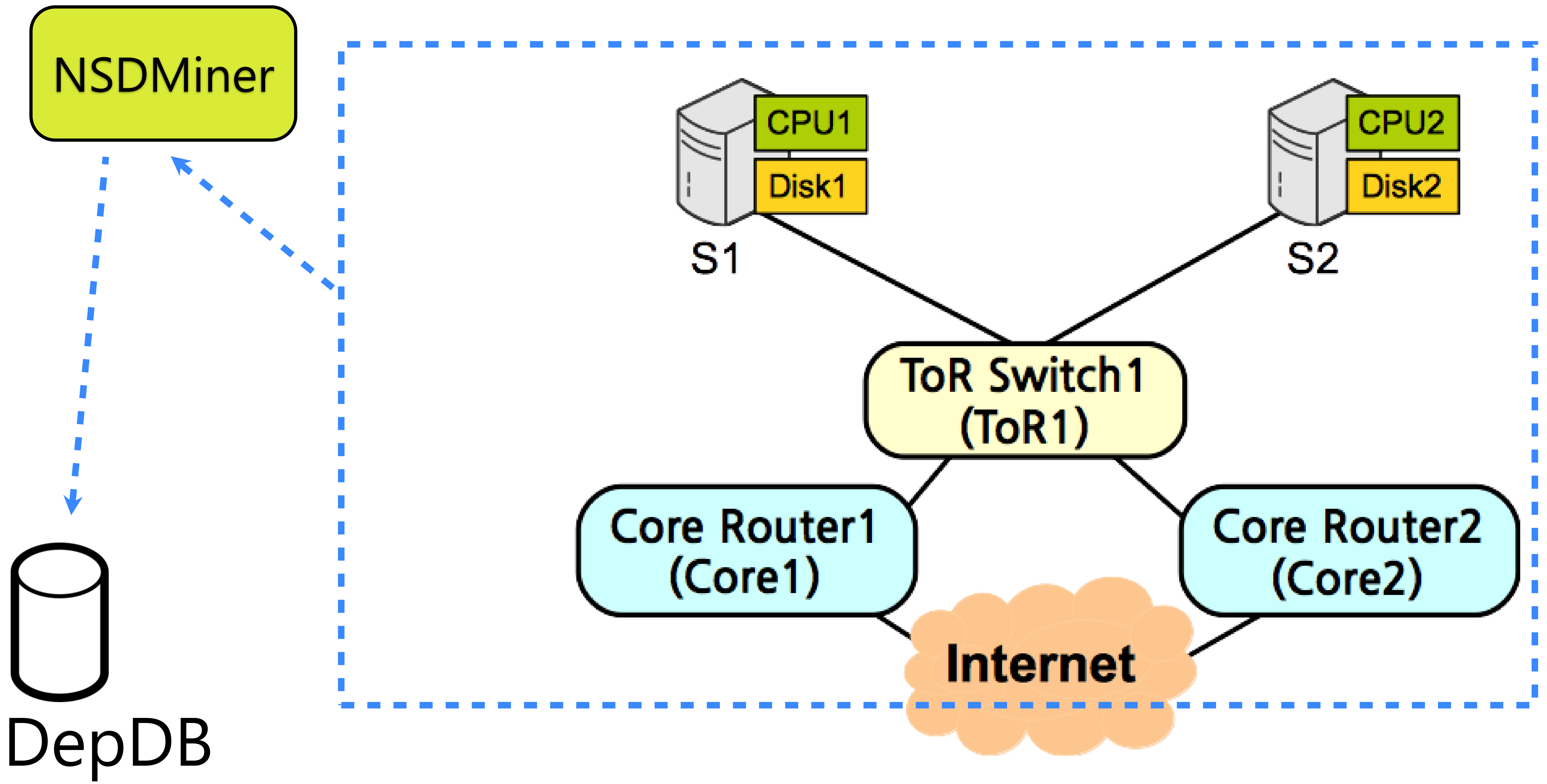


<src="S1" dst="Internet" route="ToR1,Core1"/>
<src="S1" dst="Internet" route="ToR1,Core2"/>
<src="S2" dst="Internet" route="ToR1,Core1"/>
<src="S2" dst="Internet" route="ToR1,Core2"/>
NSDMiner
CPU1
Disk1
S1
CPU2
Disk2
S2
ToR Switch1 (ToR1)
Core Router1 (Core1)
Core Router2 (Core2)
Internet

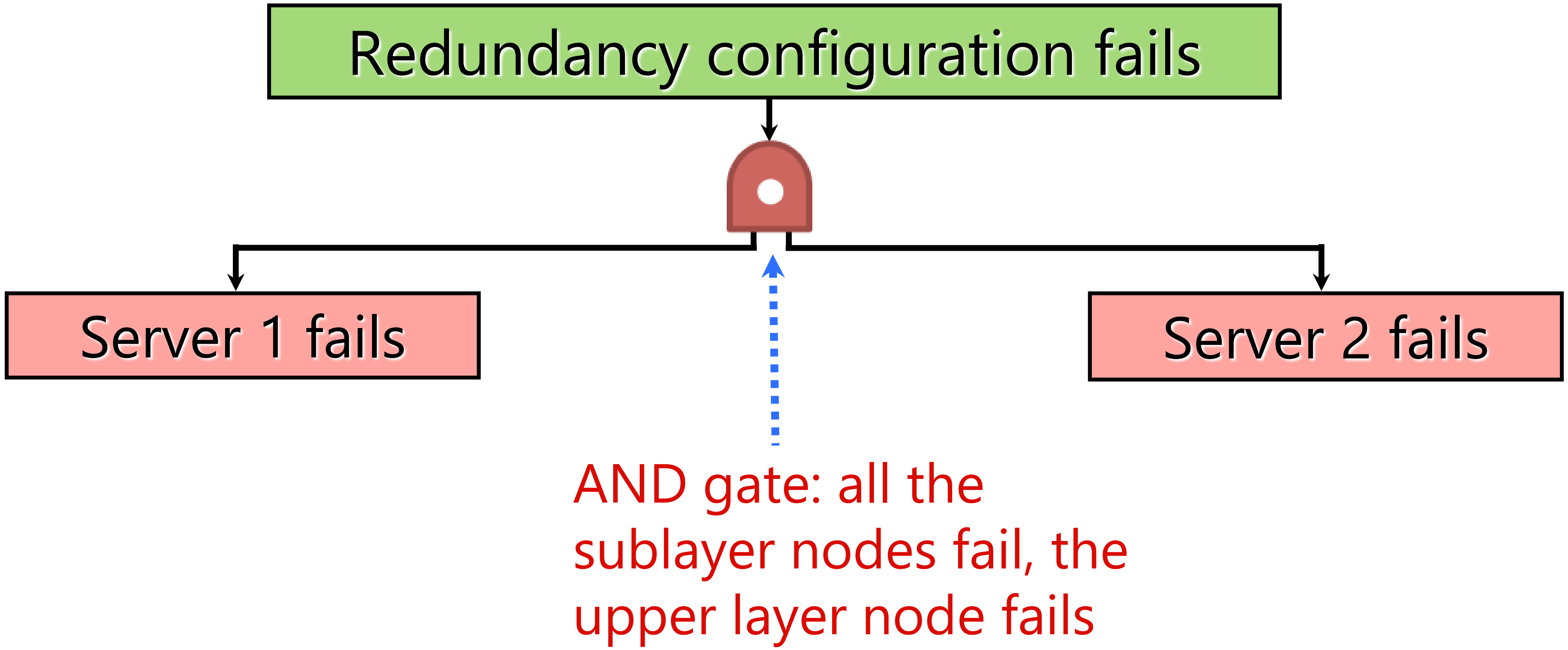
<src="S1" dst="Internet" route="ToR1,Core1"/>
<src="S1" dst="Internet" route="ToR1,Core2"/>
<src="S2" dst="Internet" route="ToR1,Core1"/>
<src="S2" dst="Internet" route="ToR1,Core2"/>

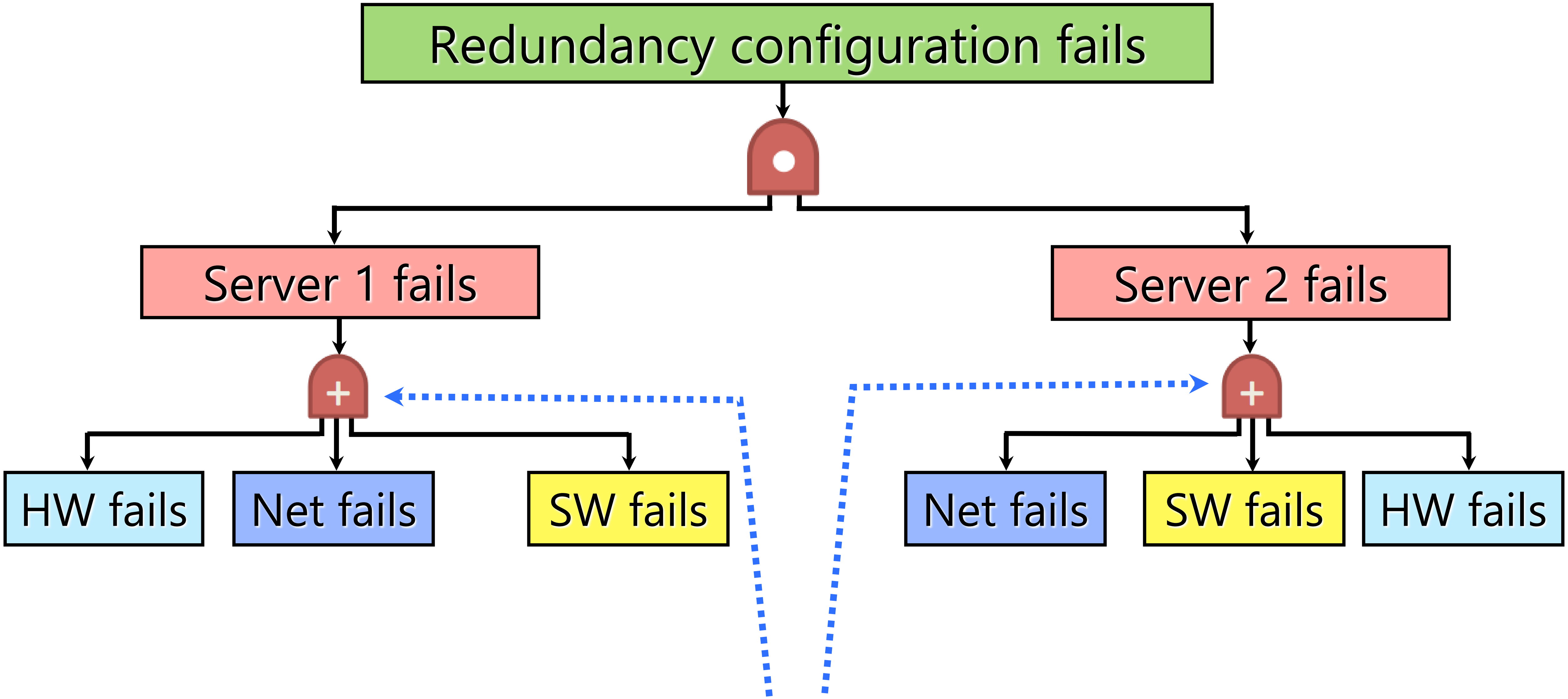


Example

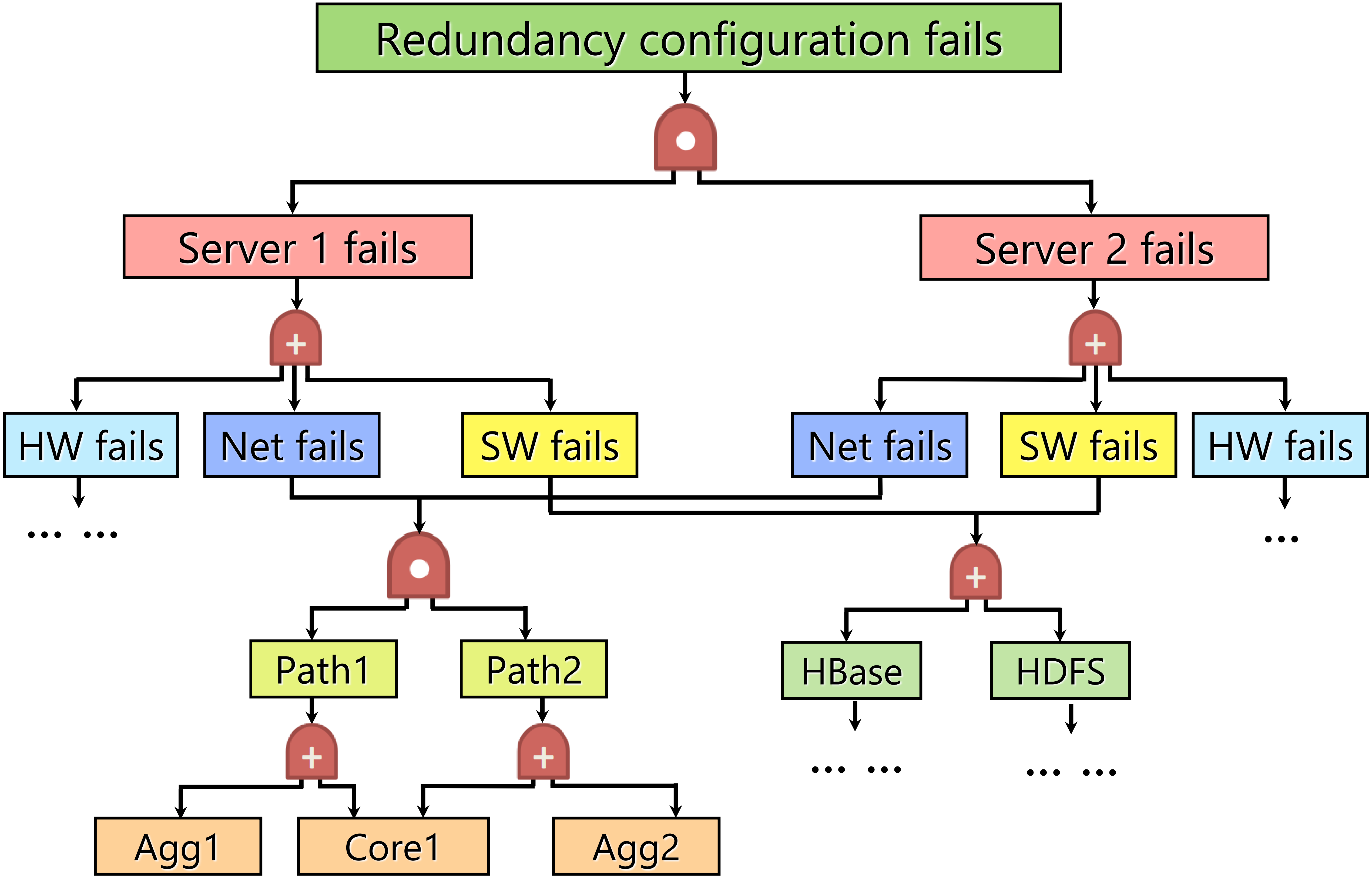

NSDMiner
DepDB
CPU1
Disk1
S1
CPU2
Disk2
S2
ToR Switch1
(ToR1)
Core Router1
(Core1)
Core Router2
(Core2)
Internet

Redundancy configuration fails

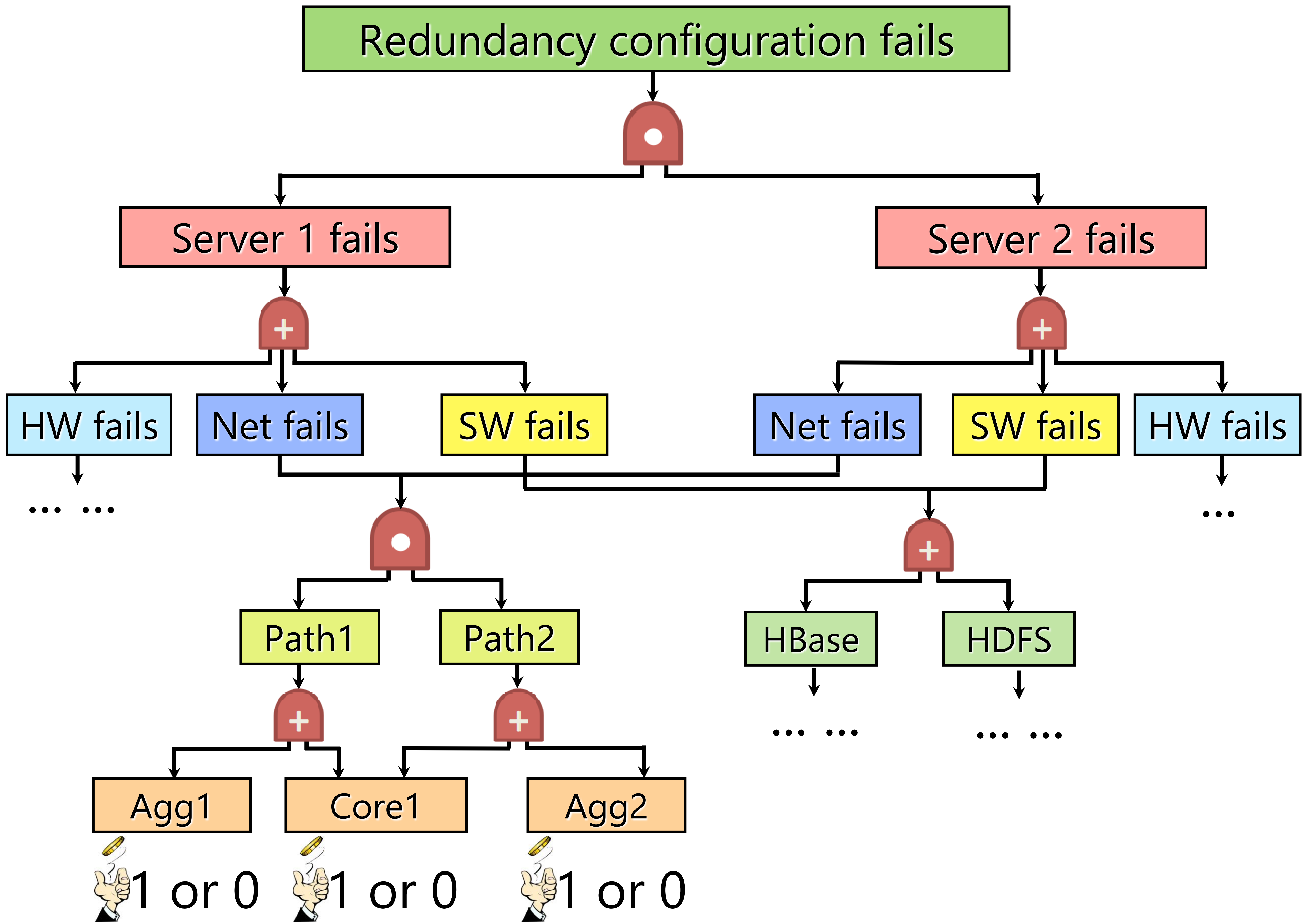

Redundancy configuration fails
Server 1 fails
Server 2 fails
AND gate: all the sublayer nodes fail, the upper layer node fails



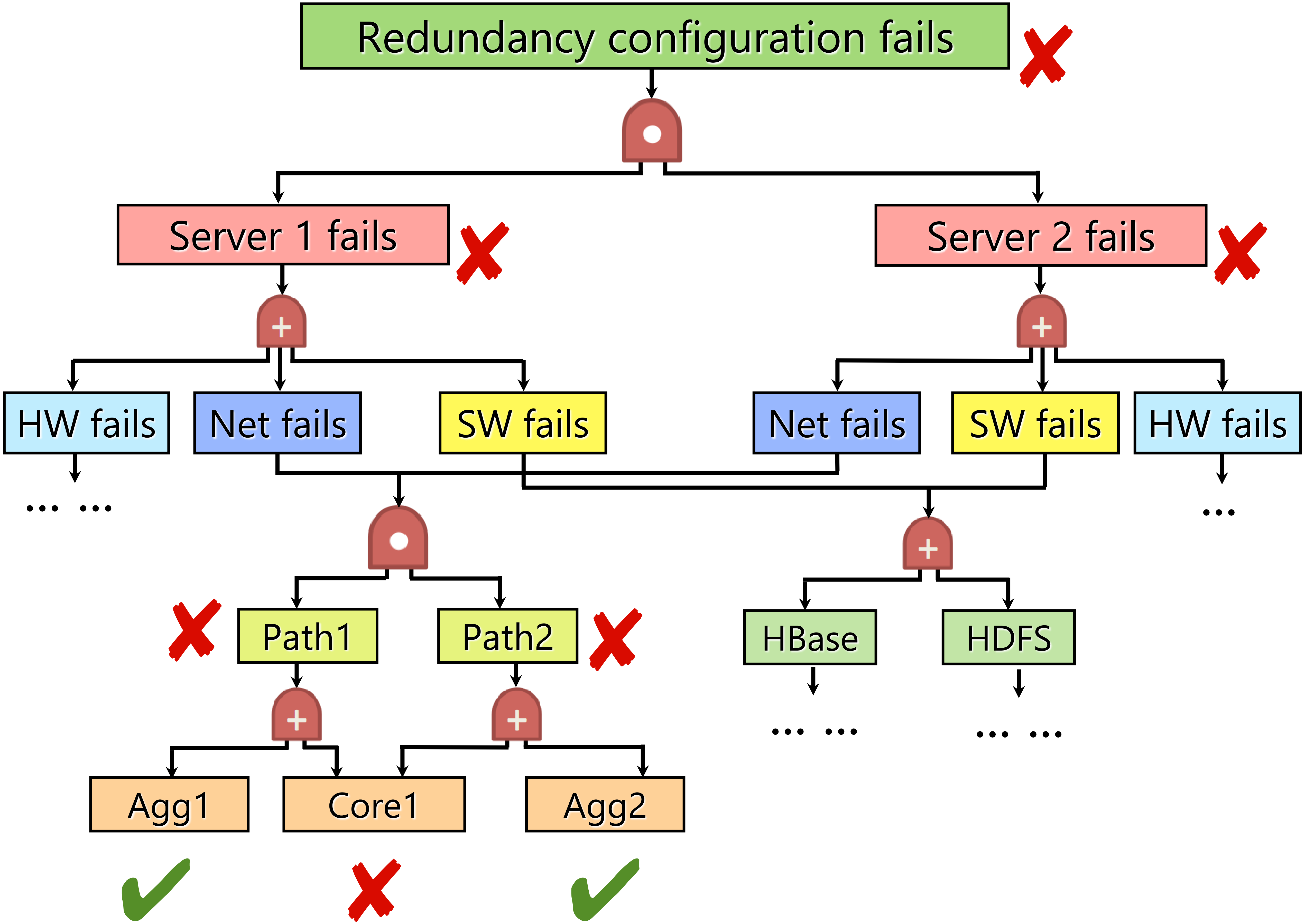
OR gate: one of the sublayer nodes fails, the upper layer node fails



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
- Fault graph analysis does not support auditing in runtime
- Cannot be used to fix the cascading failure problem

Issues in INDaaS

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

Proposed Solution:

RepAudit – An Auditing Engine

Proposed 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

- Cannot be used to fix the cascading failure problem

Proposed 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

- Cannot be used to fix the cascading failure problem

Proposed Solution: RepAudit

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

- A new domain-specific auditing language

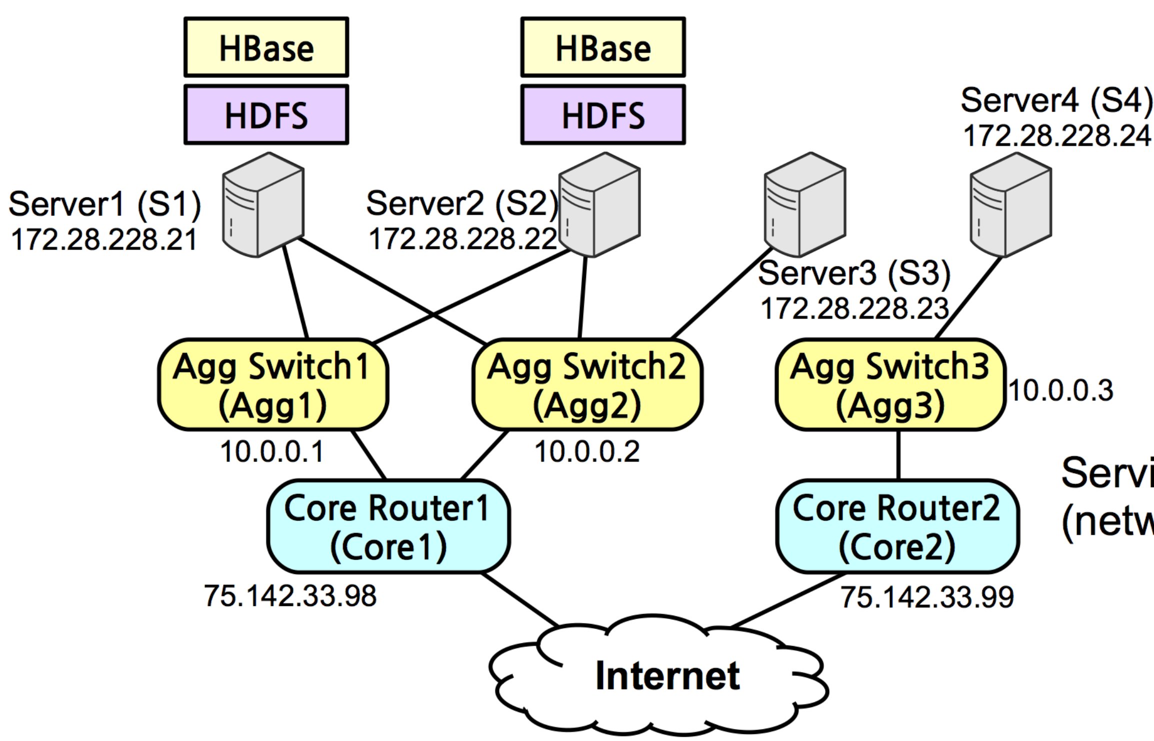
- Fault graph analysis does not support auditing in runtime

- Much faster analysis based on various SAT solvers

- Cannot be used to fix the cascading failure problem

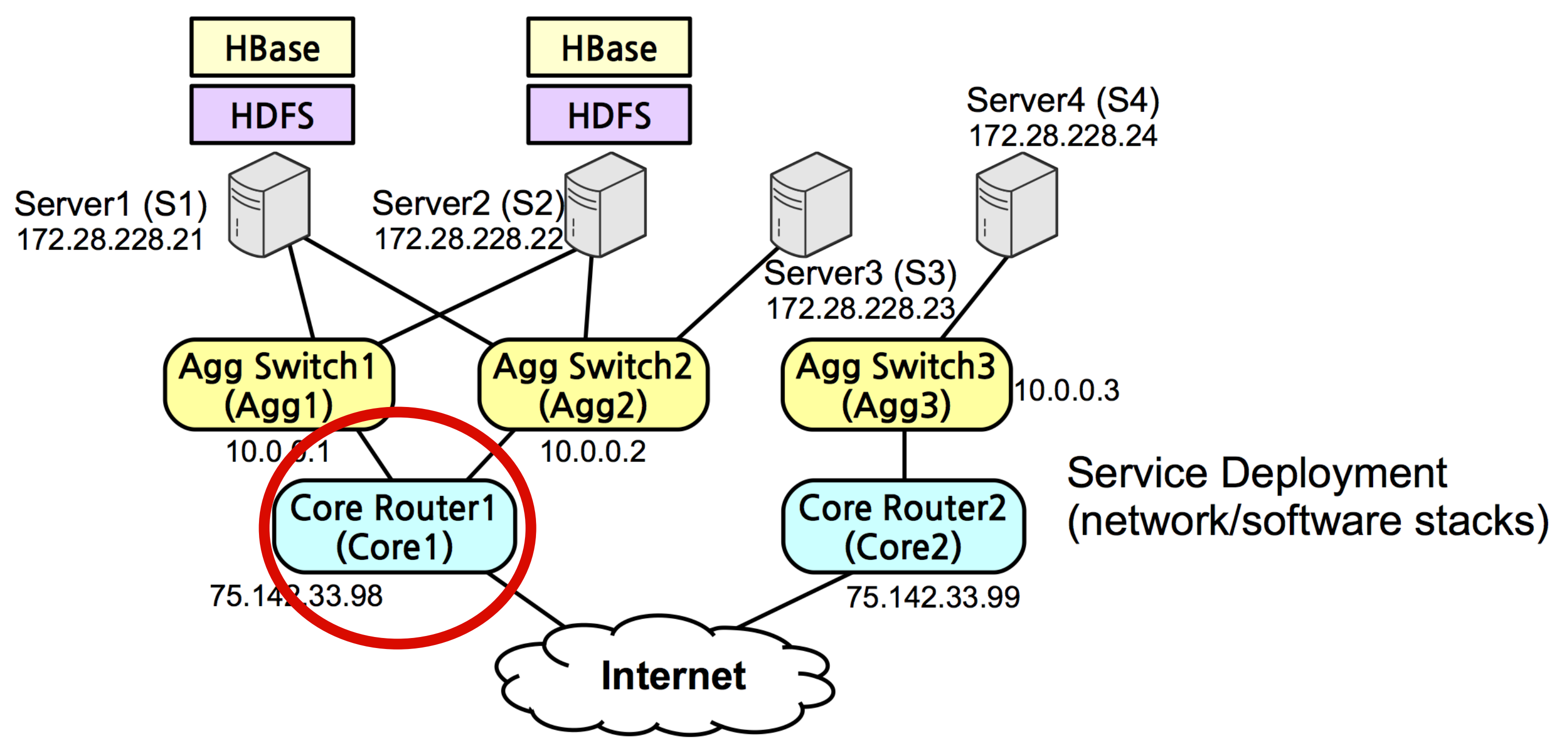
- Automatically generate improvement plans

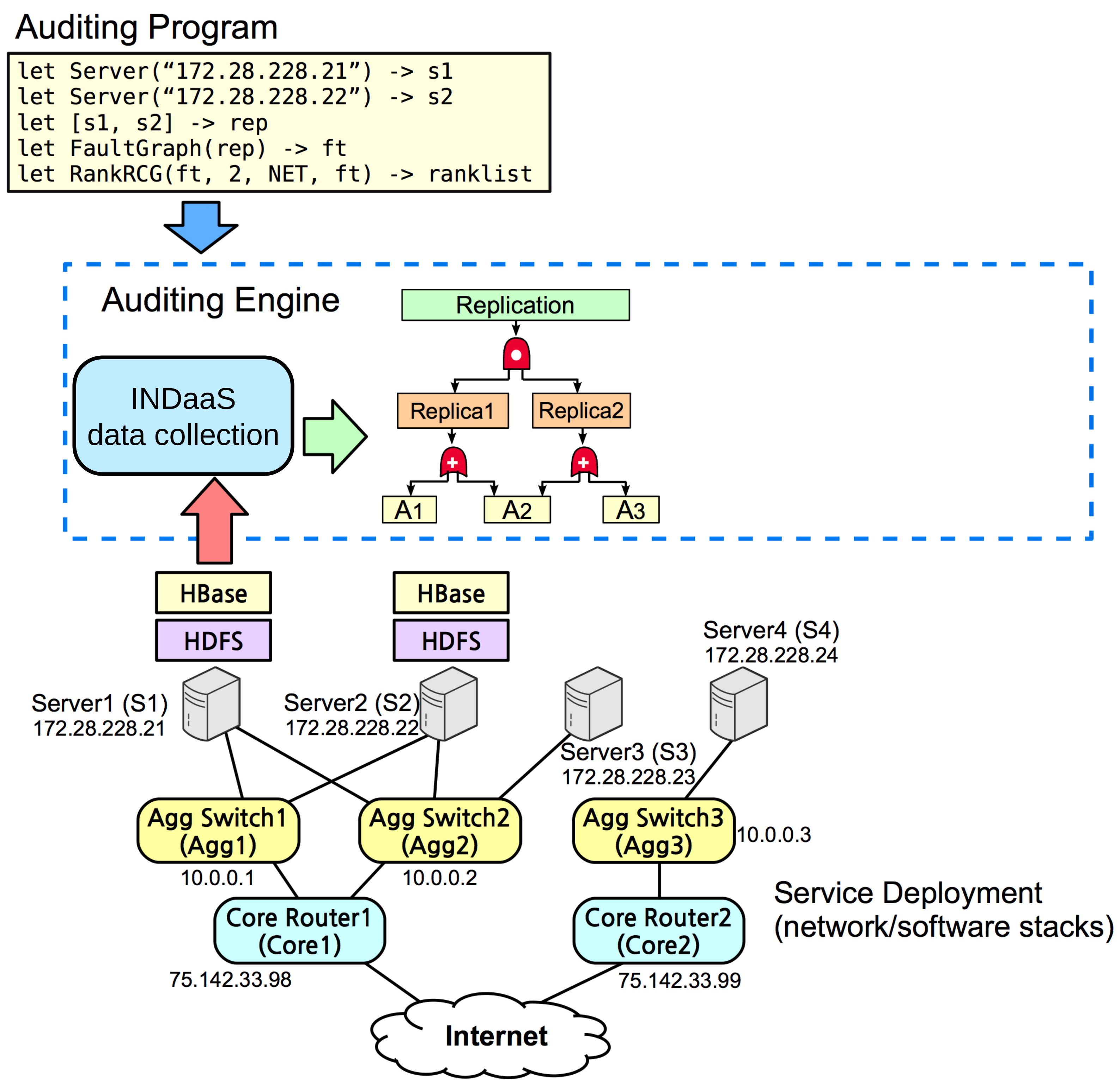
Identifying Unexpected Dependencies



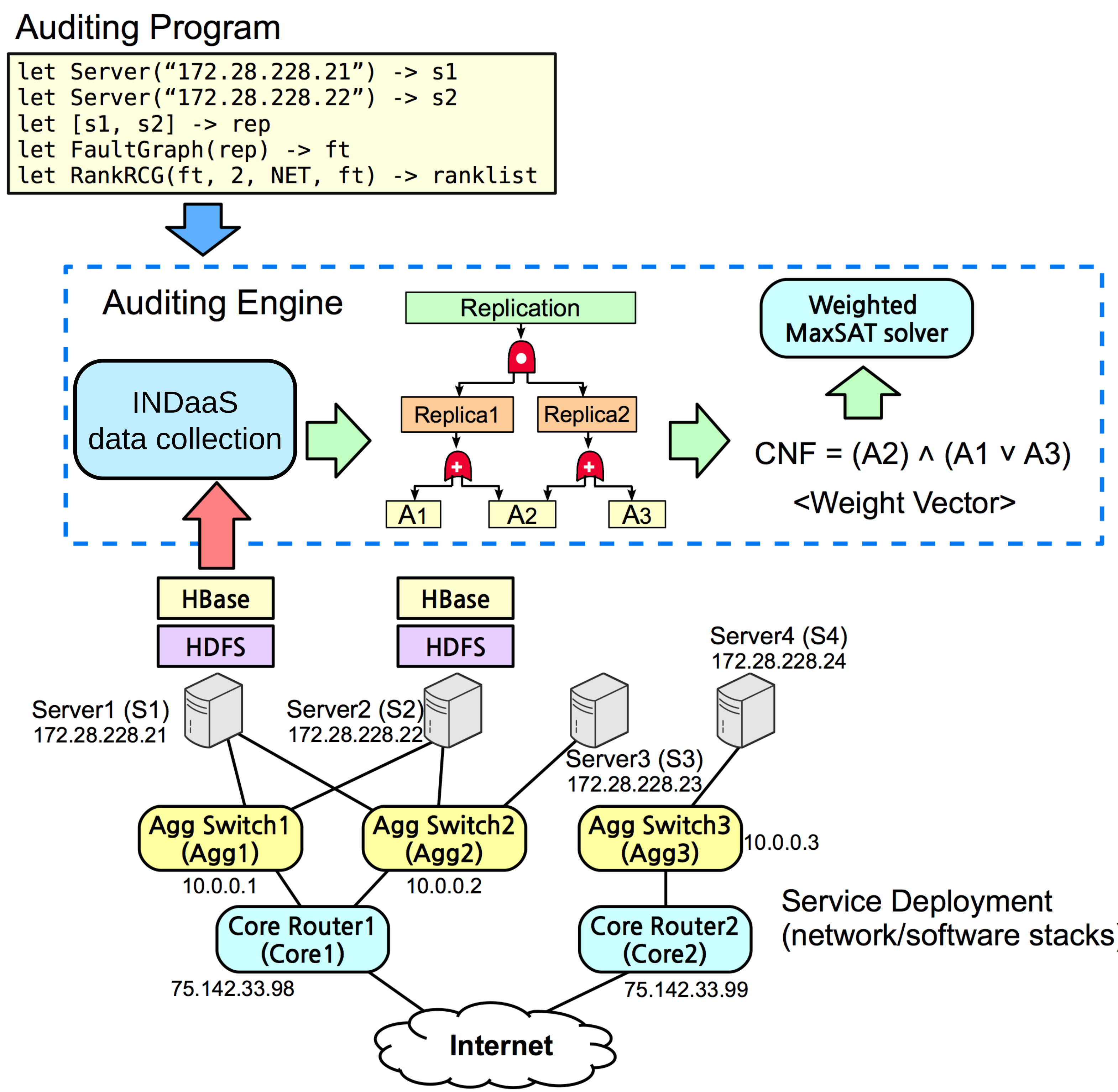
Service Deployment
(network/software stacks)

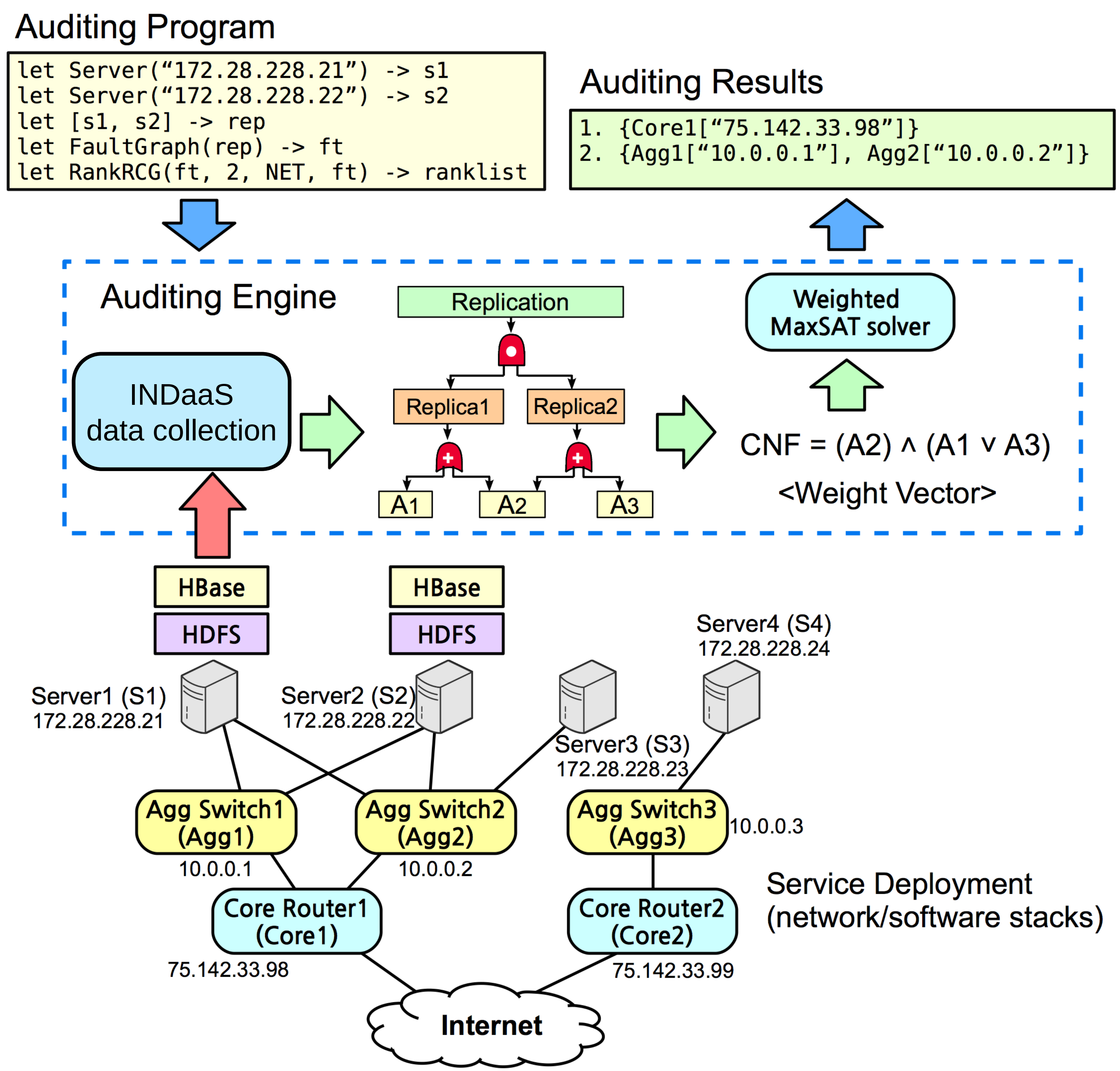
Identifying Unexpected Dependencies





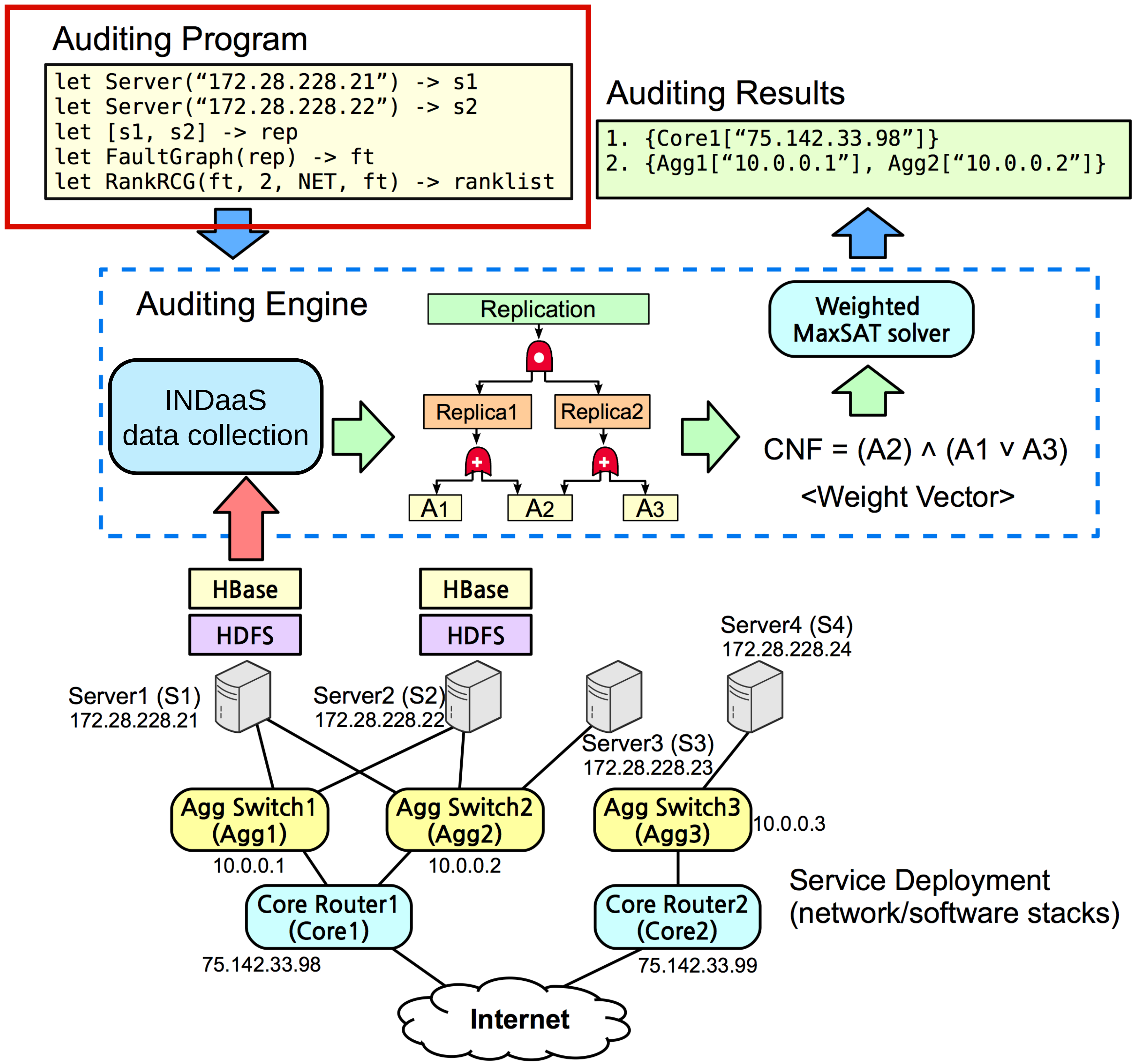
Auditing Program
let Server("172.28.228.21") -> s1
let Server("172.28.228.22") -> s2
let [s1, s2] -> rep
let FaultGraph(rep) -> ft
let RankRCG(ft, 2, NET, ft) -> ranklist
Auditing Engine
INDaaS data collection
Replication
Replica1
Replica2
A1
A2
A3
HBase
HDFS
HBase
HDFS
Server1 (S1) 172.28.228.21
Server2 (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 Engine
INDaaS data collection
Replication
Replica1
Replica2
A1
A2
A3
Weighted MaxSAT solver
CNF = (A2) ∧ (A1 ∨ A3)
<Weight Vector>
HBase
HDFS
HBase
HDFS
Server4 (S4)
172.28.228.24
Server1 (S1)
172.28.228.21
Server2 (S2)
172.28.228.22
Server3 (S3)
172.28.228.23
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
Service Deployment (network/software stacks)
Internet



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



Auditing Language

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

(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
	$\mid$	$\texttt{Switch}(e)$	Initializing switch node
	$\mid$	$\texttt{FaultGraph}(e)$	Generating fault graph
	$\mid$	$\texttt{RankRCG}(e_1,e_2,m,t)$	Ranking RCGs
	$\mid$	$\texttt{RankNode}(e,m,t)$	Ranking devices
	$\mid$	$\texttt{FailProb}(e,t)$	Failure probability
	$\mid$	$\texttt{RecRep}(e_1,e_2,m)$	Recommendation
	$\mid$	$\cdots$	
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

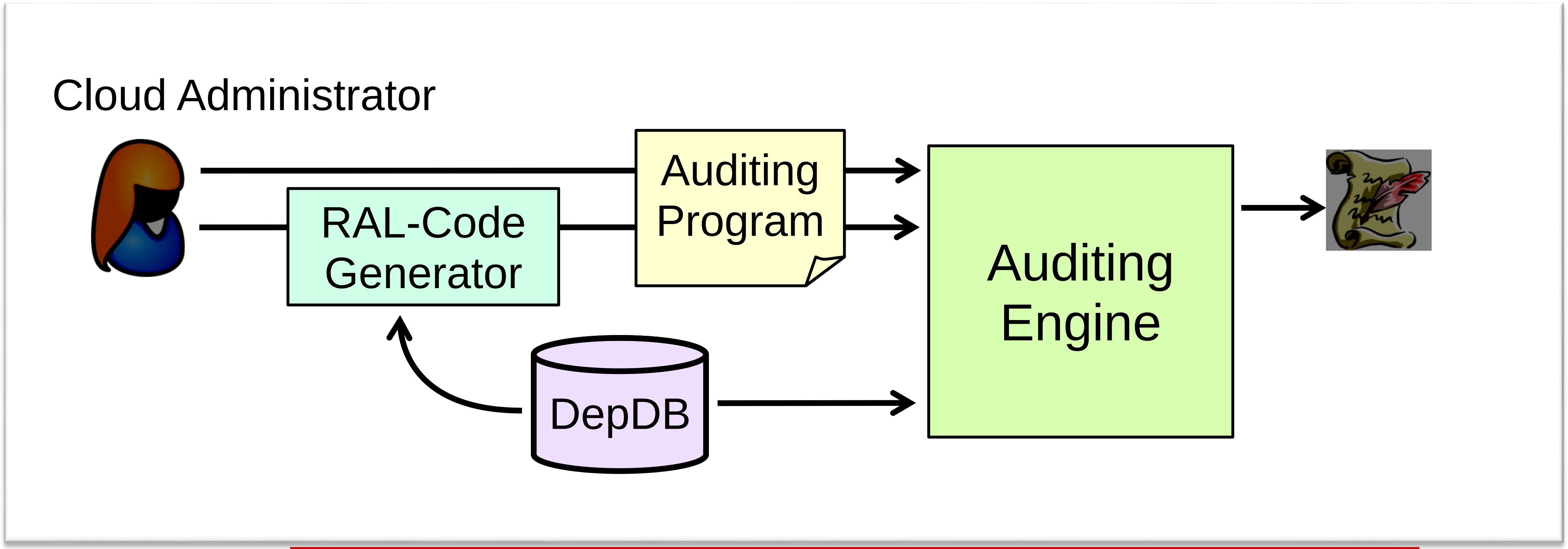
S	$::=$	$\texttt{let}\ e \rightarrow g$	Assignment
	$\mid$	$\texttt{print}(e)$	Output
	$\mid$	$S_1;S_2 \mid \texttt{if}(e)\{S_1\}\ \texttt{else}\{S_2\} \mid \texttt{while}(e)\{S\}$	

(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
	$\mid$	$\texttt{Switch}(e)$	Initializing switch node
	$\mid$	$\texttt{FaultGraph}(e)$	Generating fault graph
	$\mid$	$\texttt{RankRCG}(e_1,e_2,m,t)$	Ranking RCGs
	$\mid$	$\texttt{RankNode}(e,m,t)$	Ranking devices
	$\mid$	$\texttt{FailProb}(e,t)$	Failure probability
	$\mid$	$\texttt{RecRep}(e_1,e_2,m)$	Recommendation
	$\mid$	...	
m	$::=$	$\texttt{SIZE} \mid \texttt{PROB}$	Ranking metric
t	$::=$	$\texttt{NET} \mid \texttt{SoftW} \mid \texttt{HardW}$	Dependency types

(b) Expressions of RAL.

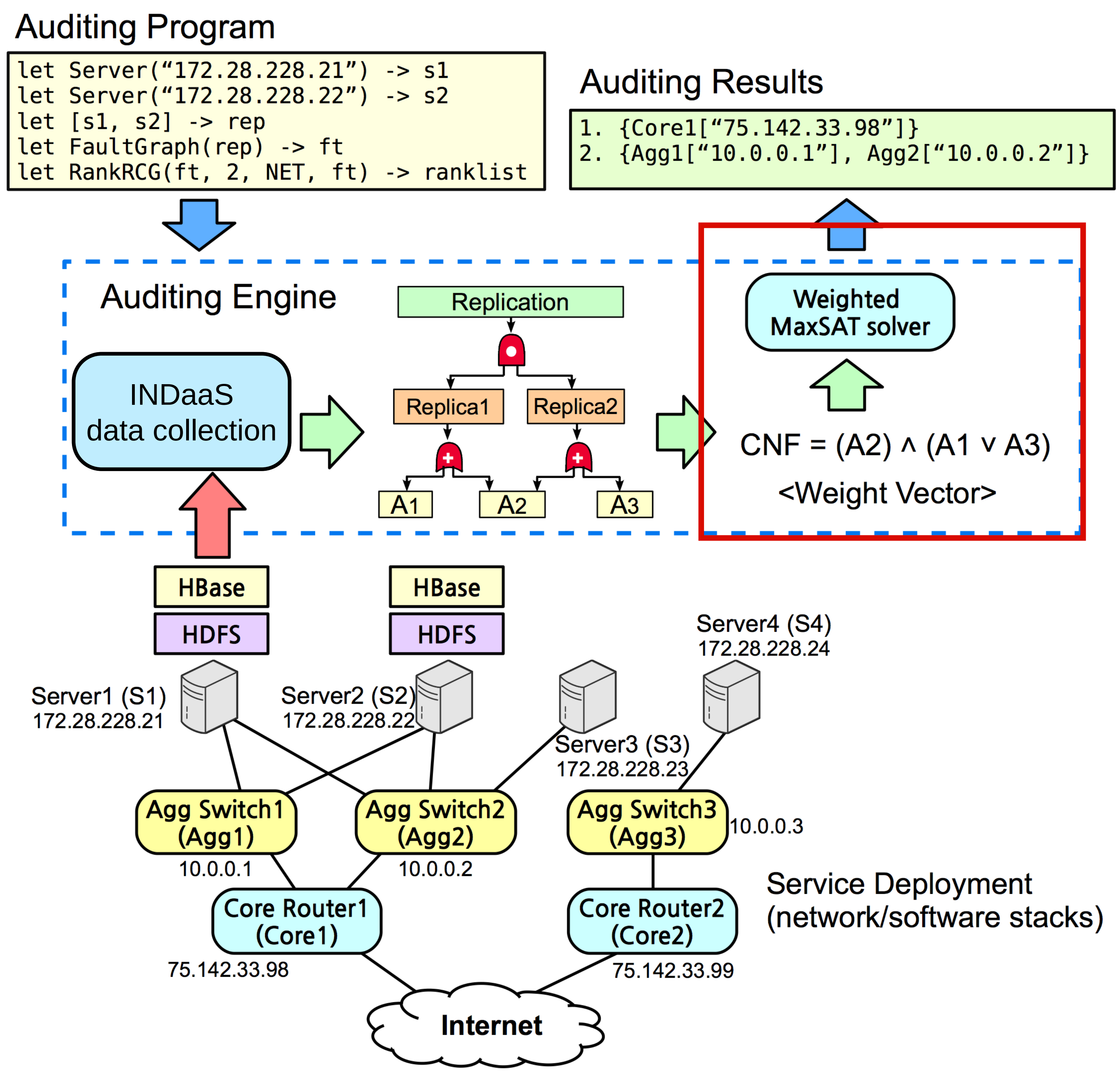
Auditing Language



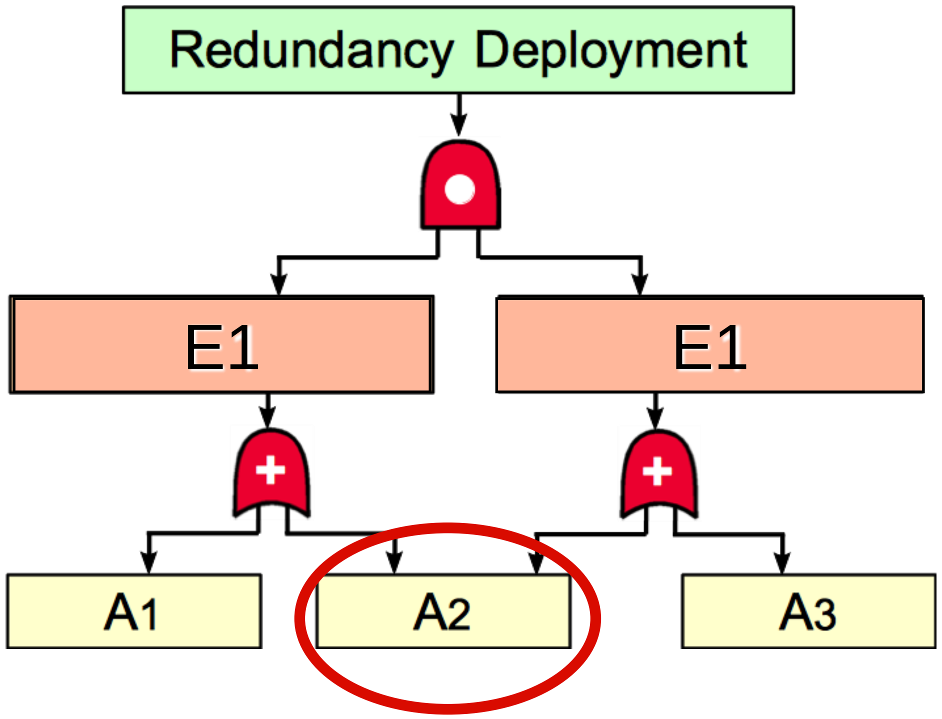
q	::=	Server(e)	Initializing server node
	|	Switch(e)	Initializing switch node
	|	FaultGraph(e)	Generating fault graph
	|	RankRCG(e_1, e_2, m, t)	Ranking RCGs
	|	RankNode(e, m, t)	Ranking devices
	|	FailProb(e, t)	Failure probability
	|	RecRep(e_1, e_2, m)	Recommendation
	|	...	
m	::=	SIZE | PROB	Ranking metric
t	::=	NET | SoftW | HardW	Dependency types

(b) Expressions of RAL.

RepAudit's Contributions

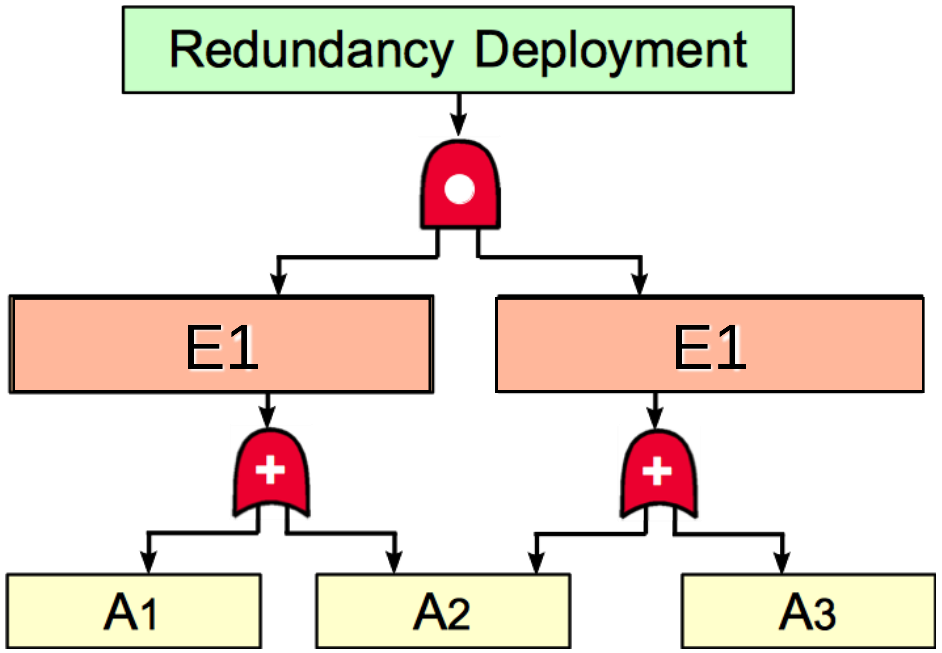


Risk Groups in Fault Graphs



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

Risk Groups in Fault Graphs

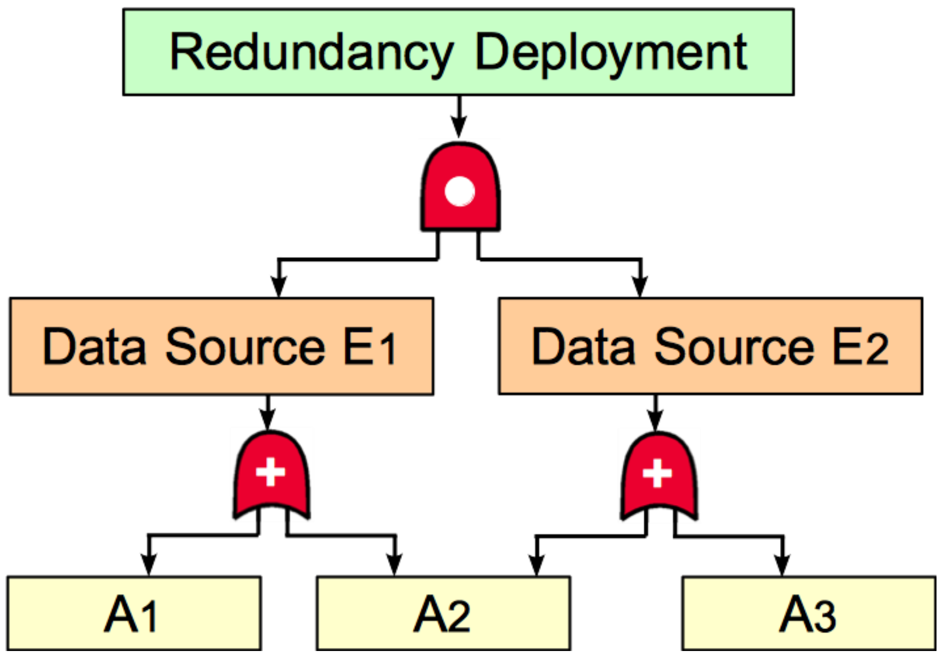


A risk group means a set of leaf nodes whose simultaneous failures lead to the failure of root node.

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

{A1} or {A3} is not risk group

Reducing the Problem to SAT



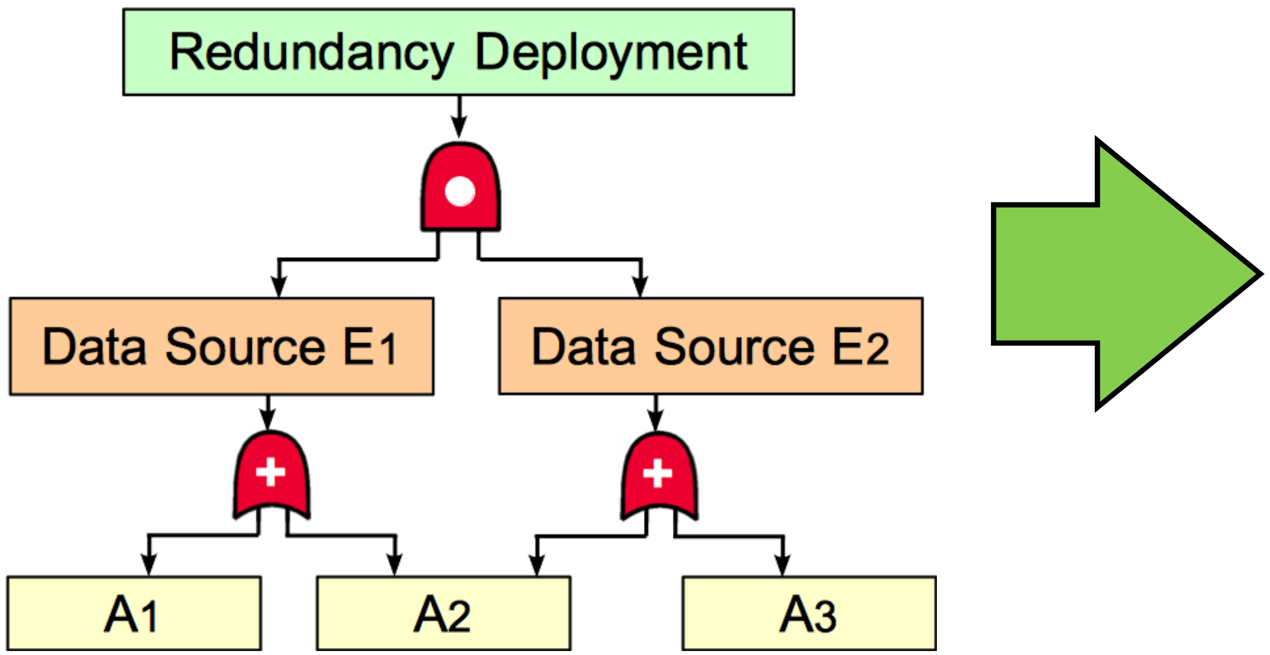
Boolean formula

$= E_1 \wedge E_2$

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

- Extracting risk groups can be reduced to the problem of finding satisfying assignments for a Boolean formula
- E.g., {A1=0, A2=1, A3=0} represents a risk group

Reducing the Problem to SAT



Boolean formula

$= E_1 \wedge E_2$

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

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

Min-cost SAT Problem

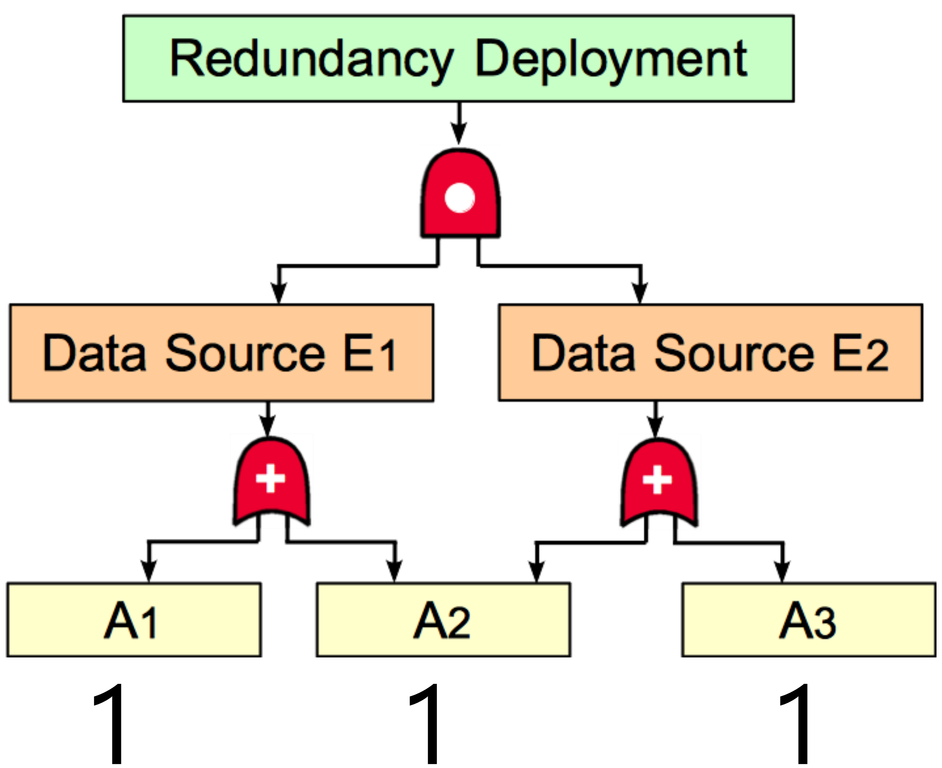
For a given Boolean formula φ with n variables $x_1, x_2,..., x_n$, and a corresponding cost vector, $\{c_i \mid c_i \geq 0, 1 \leq i \leq n\}$, the goal is to find a satisfying assignment for φ that minimizes the formula:

$$C = \sum_{i=1}^{n} c_i x_i$$

- To find ranking by size we use $c_i = 1$
- If we know the failure probability of each component, we can compute ranking by failure probability

Discovering Risk Groups

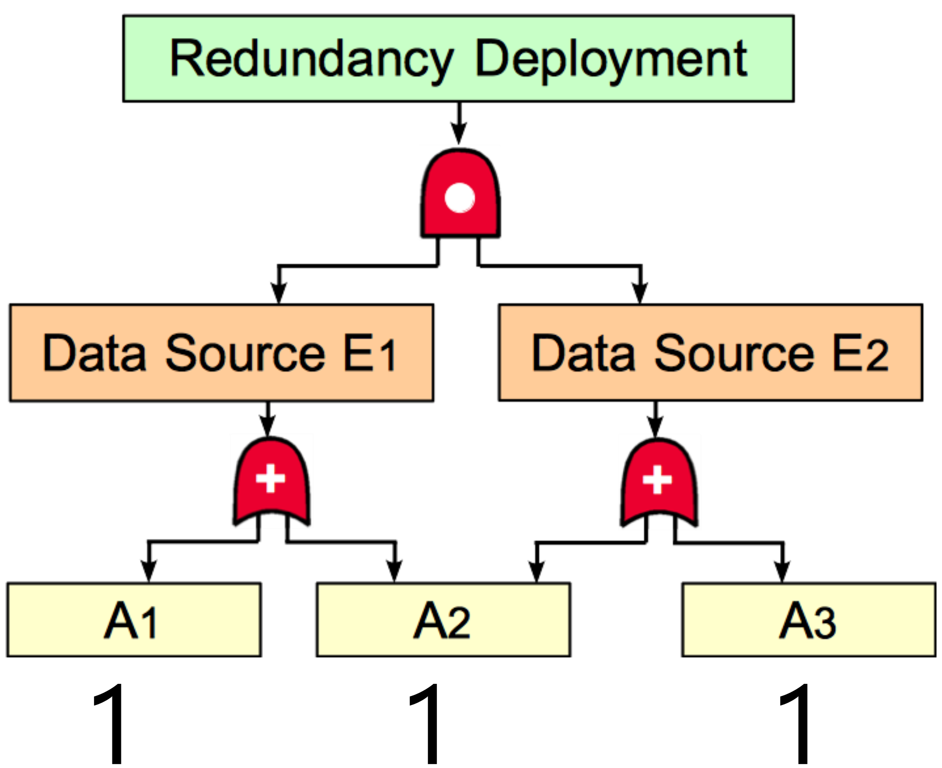
- Using weighted MaxSAT solver
  - Satisfiable assignment with the least weights
  - Obtain the least $C = \sum c_i \cdot w_i$
  - Very fast with 100% accuracy



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

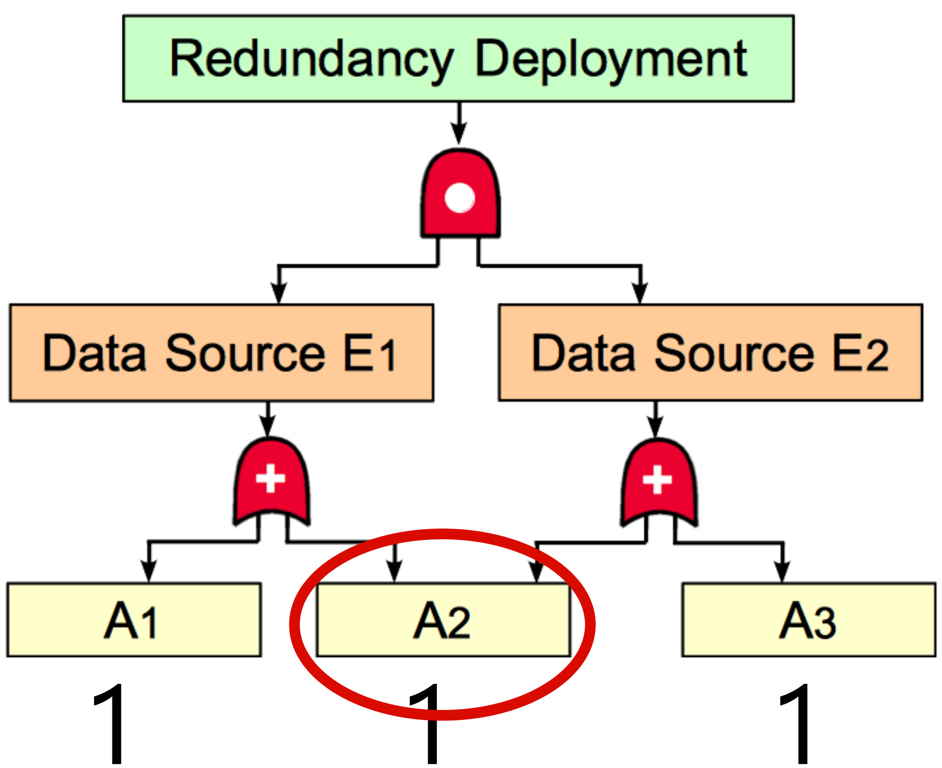
- Using weighted MaxSAT solver
  - Satisfiable assignment with the least weights
  - Obtain the least $C = \sum c_i \cdot w_i$
  - Very fast with 100% accuracy



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

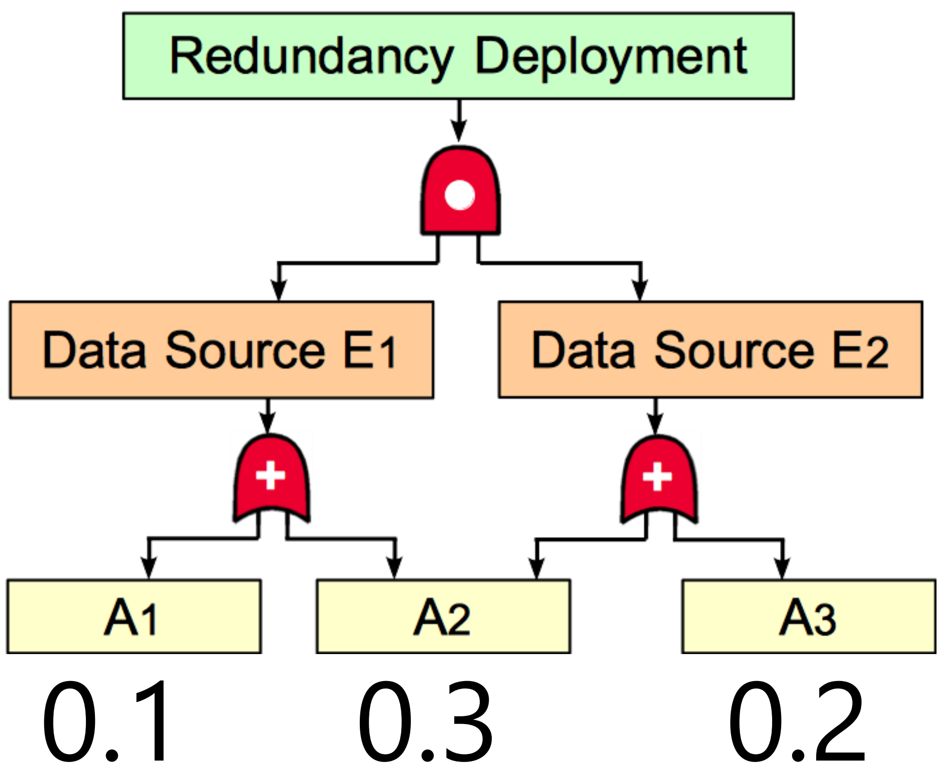
- Using weighted MaxSAT solver
  - Satisfiable assignment with the least weights
  - Obtain the least $C = \sum c_i \cdot w_i$
  - Very fast with 100% accuracy



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 top-k critical Risk Groups by Failure Probability

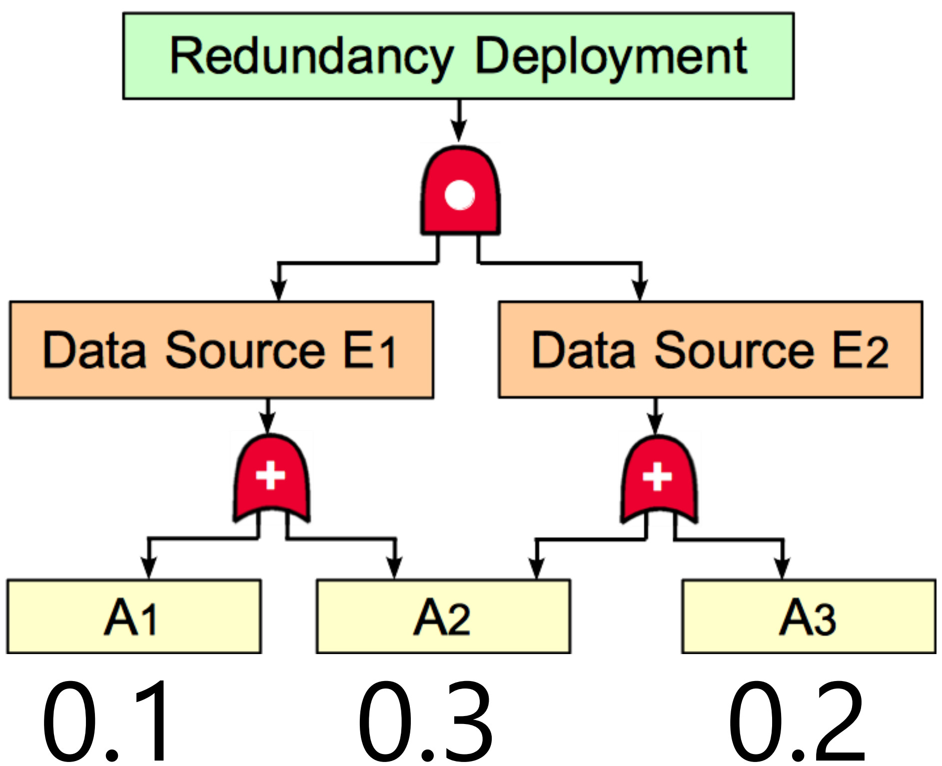
- If we can obtain failure probability of each component:



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

Discovering top-k critical Risk Groups by Failure Probability

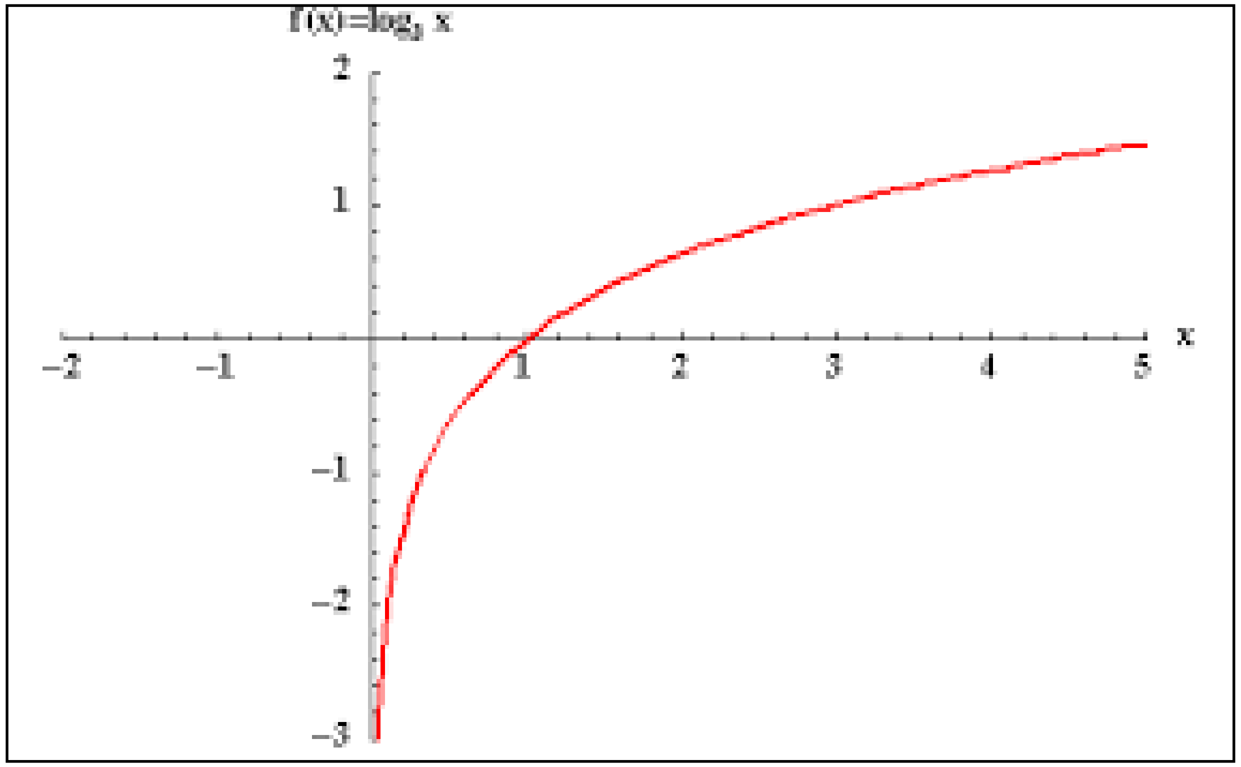
- If we can obtain failure probability of each component:



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

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

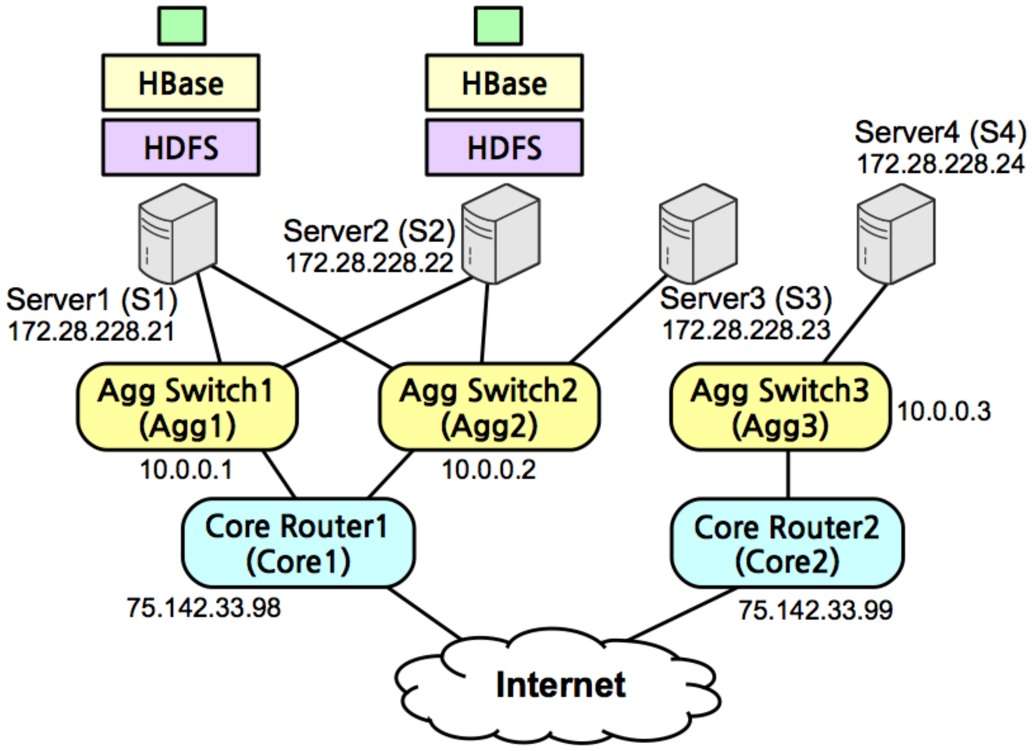


Discovering top-k critical Risk Groups

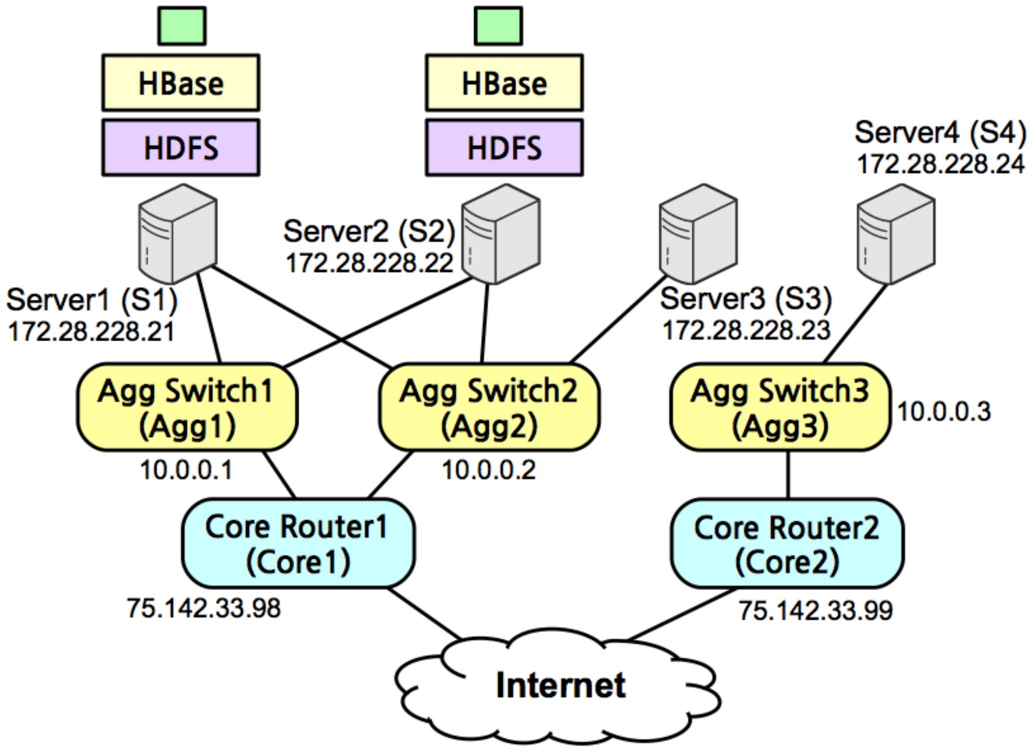
- Find out the top-k critical risk groups through *k* loop iterations
  - Use a ∧ to connect the current formula and the negation of the found assignment

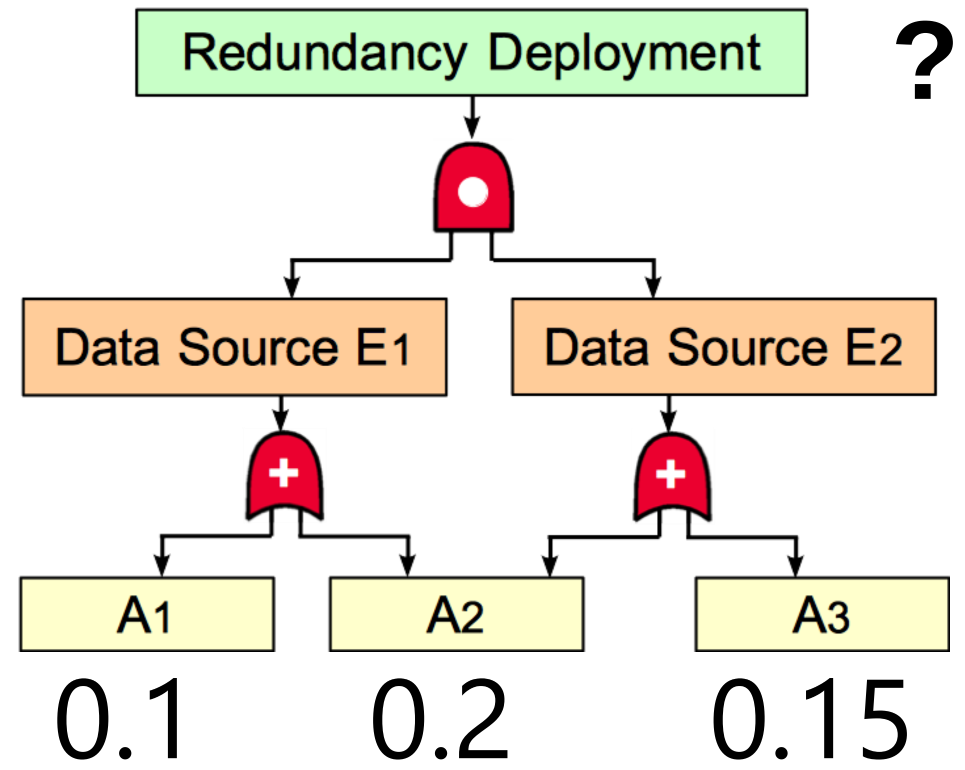
$$(A_1 \vee A_2) \wedge (A_2 \vee A_3) \wedge \neg(\neg A_1 \wedge A_2 \wedge \neg A_3)$$

RAL Primitive: Failure Probability

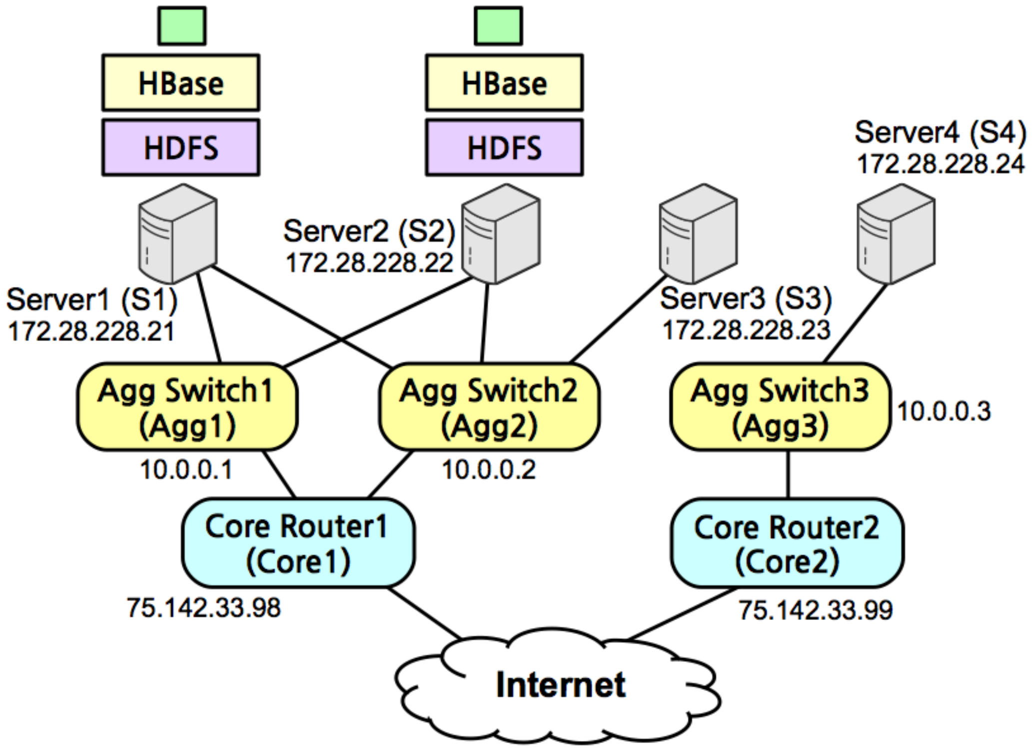


RAL Primitive: Failure Probability



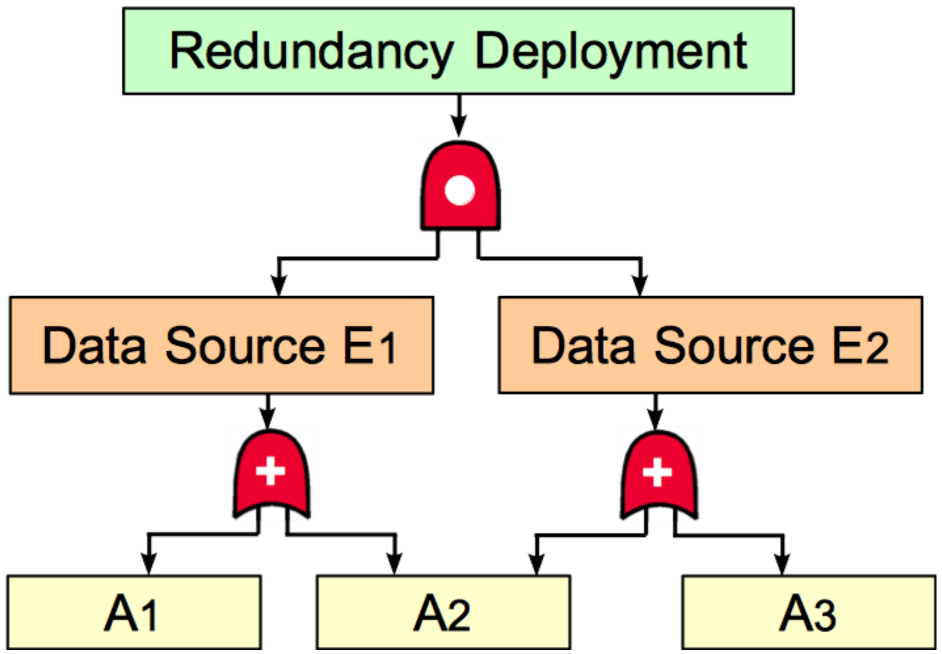


Example: Failure Probability




```
let Server("172.28.228.21") -> s1;
let Server("172.28.228.22") -> s2;
let [s1. s2] -> rep;
let FaultGraph(rep) -> ft;
let FailProb(ft, NET) -> prob;
print(prob);
```

Model Counter Example

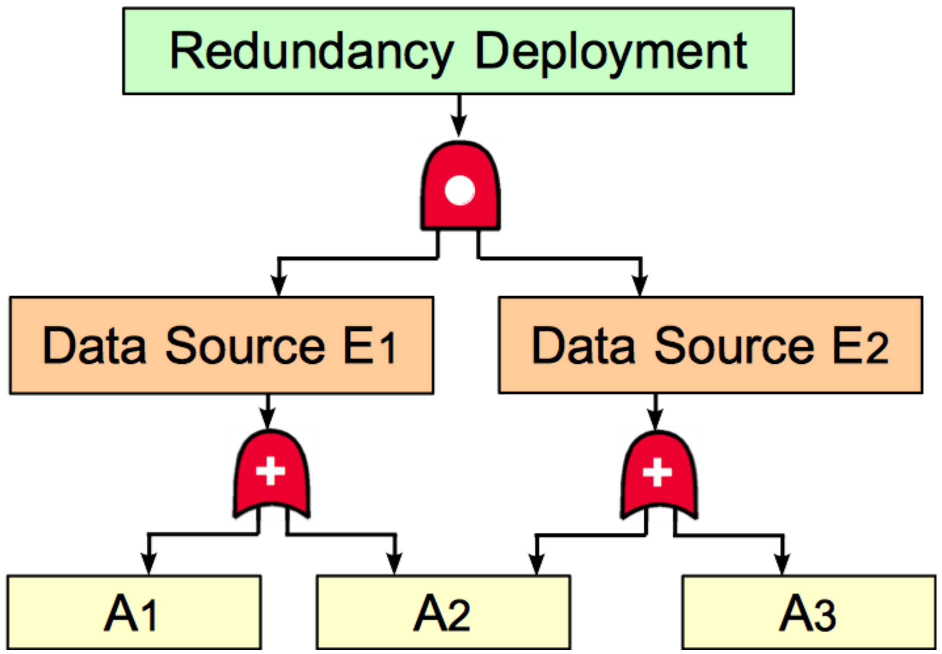


If we assume the failure probability of each leaf node is 0.5

$(A_1 \vee A_2) \wedge (A_2 \vee A_3)$ → Model Counter → 5

Failure probability =5 /(2^3) = 5/8

Model Counter Example

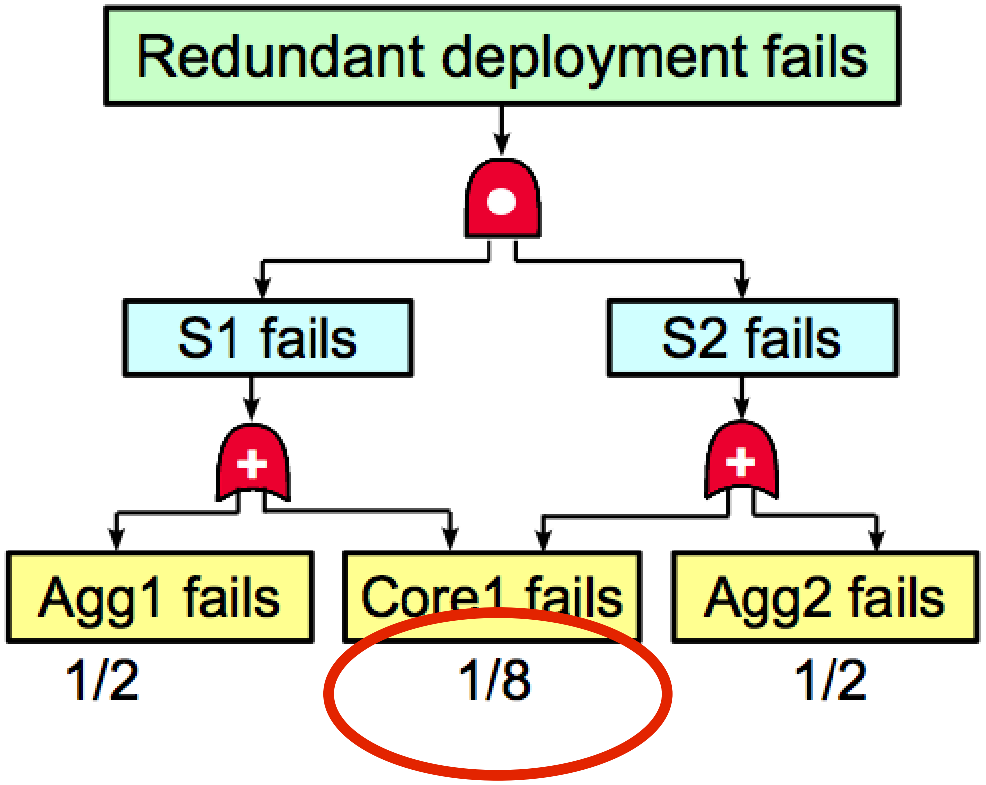


The probability of Leaf nodes is not 0.5 in practice.

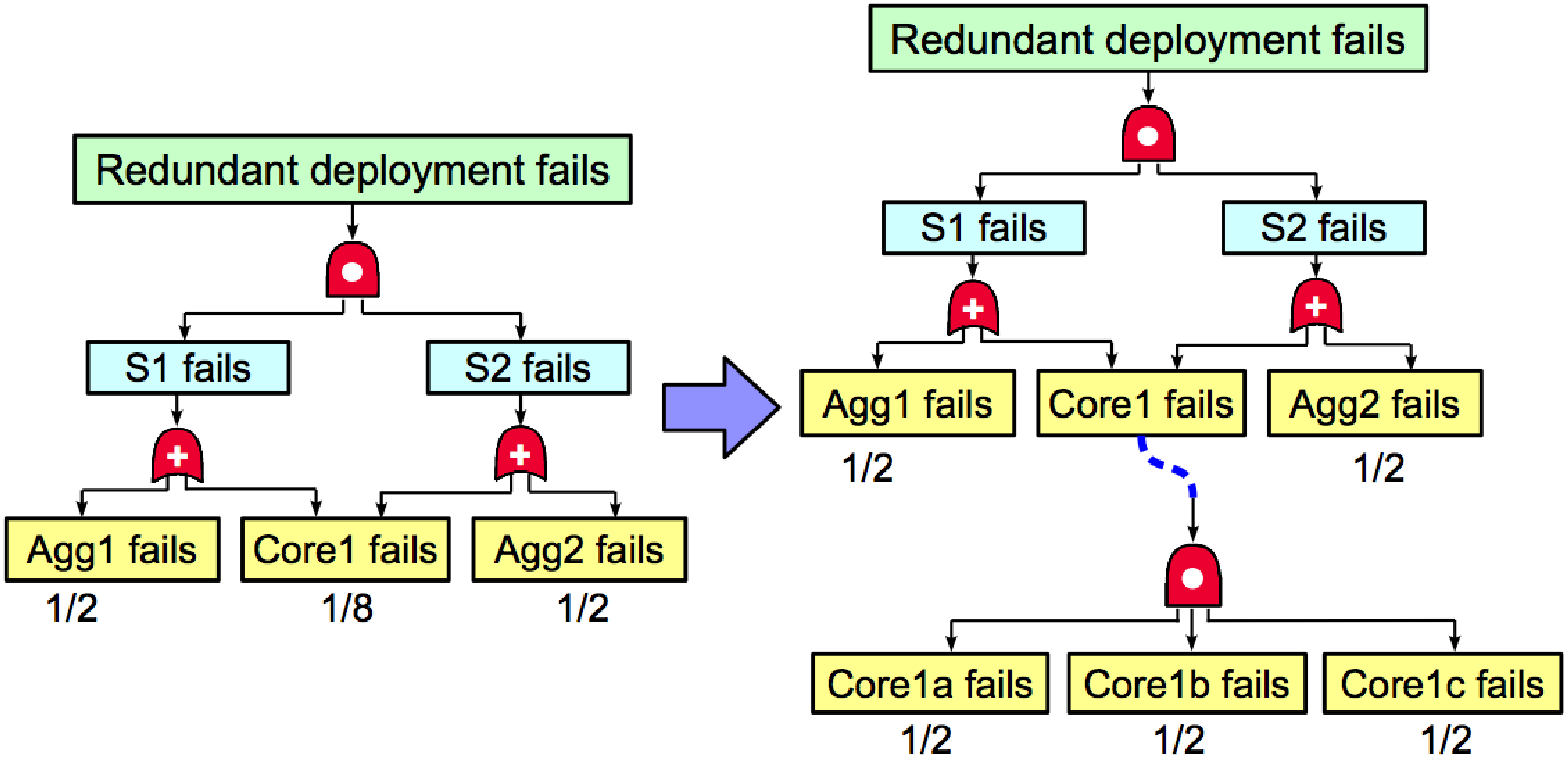
$(A_1 \vee A_2) \wedge (A_2 \vee A_3)$ → Model Counter → 5

Failure probability =5 /(2^3) = 5/8

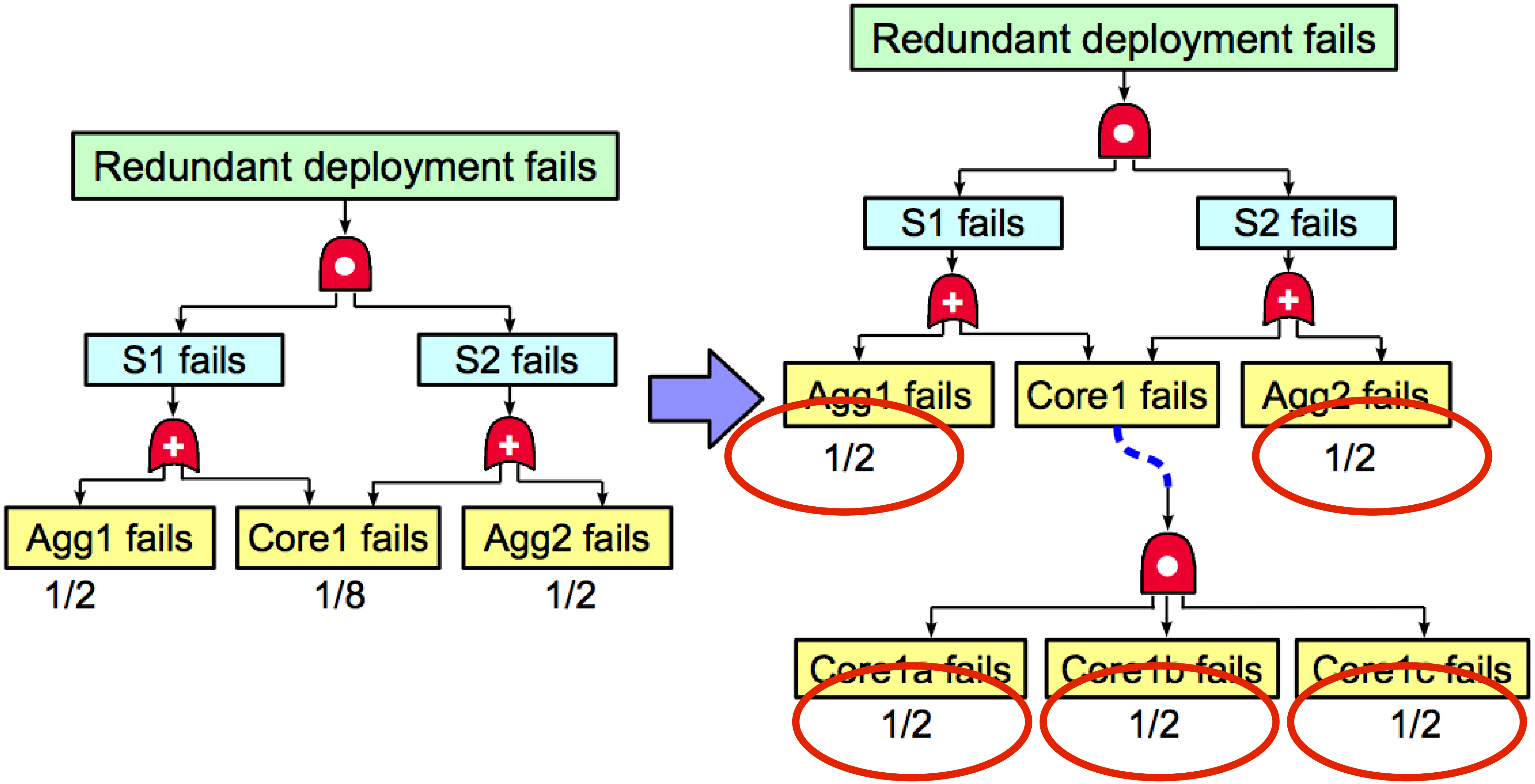
Model Counter Example



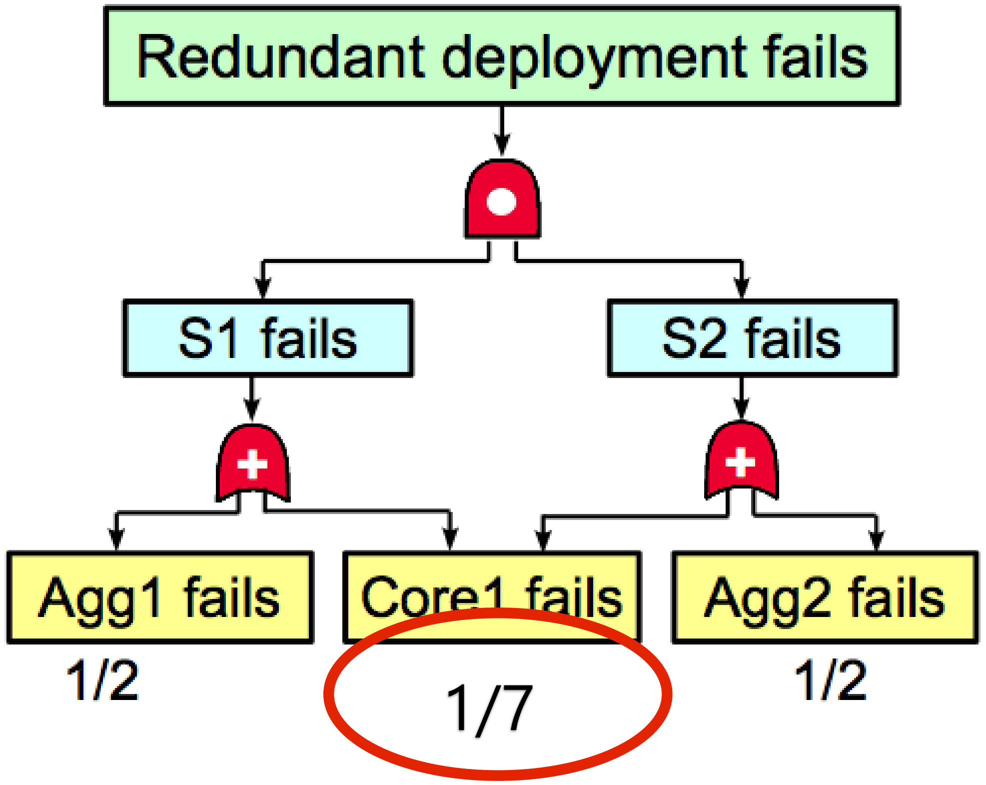
Model Counter Example



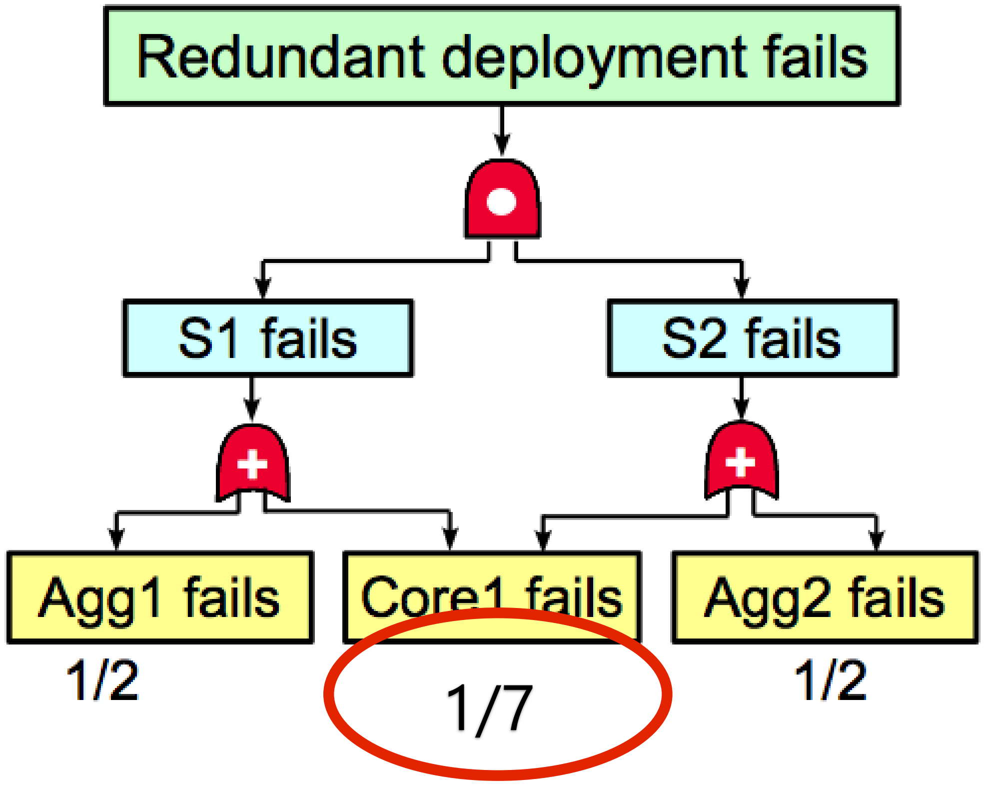
Model Counter Example



Model Counter Example



Model Counter Example



We use an approximate algorithm

$1/7 \approx 1/8 + 1/64 + 1/512$

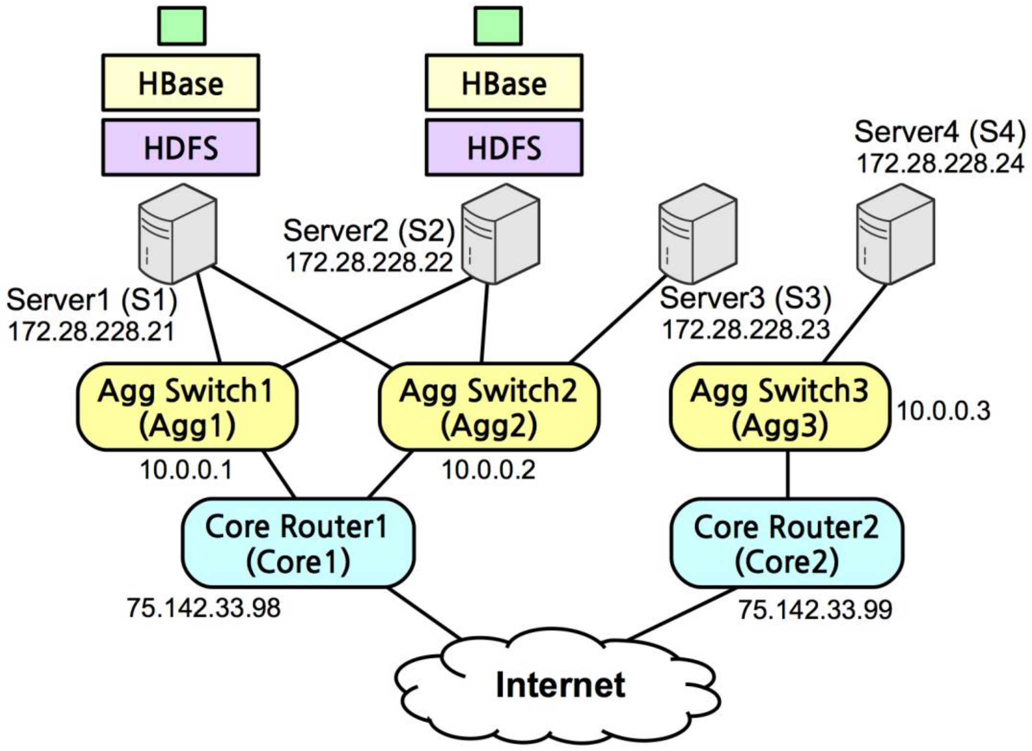
Proposed Solution: RepAudit

- 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
- Cannot be used to fix the cascading failure problem
  - Automatically generate improvement plans

Proposed Solution: RepAudit

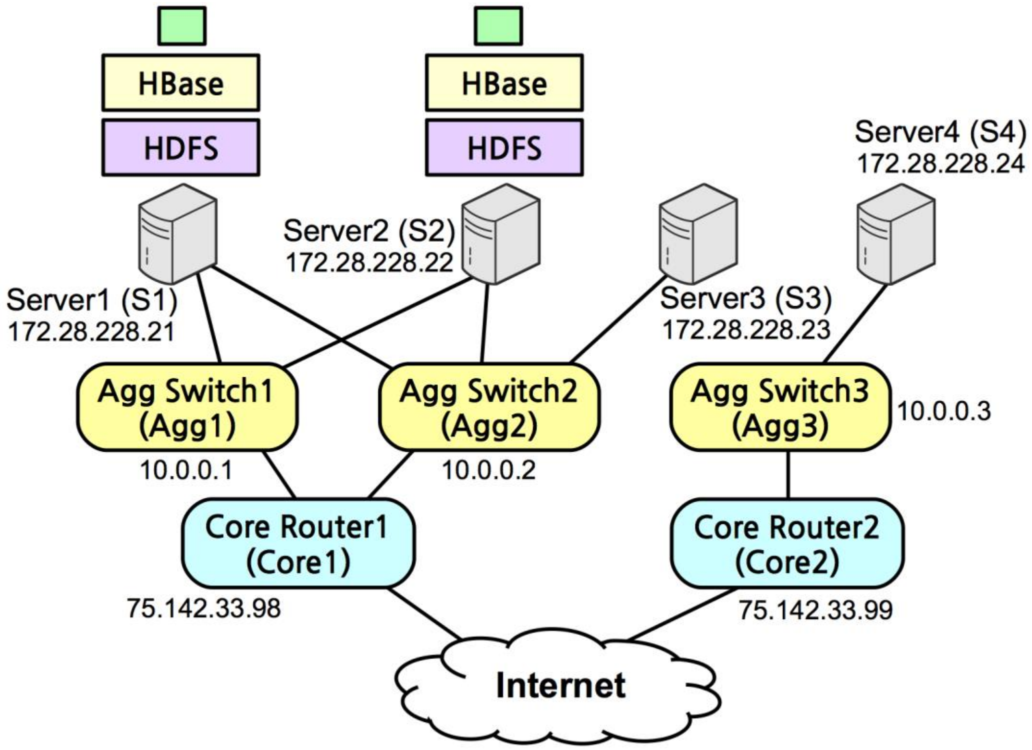
- Hard to express diverse auditing tasks
  - A new domain-specific auditing language
- Fault graph analysis does not support auditing in runtime
  - Much faster analysis based on SAT solver variants
- Cannot be used to fix the cascading failure problem
  - Automatically generate improvement plans

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

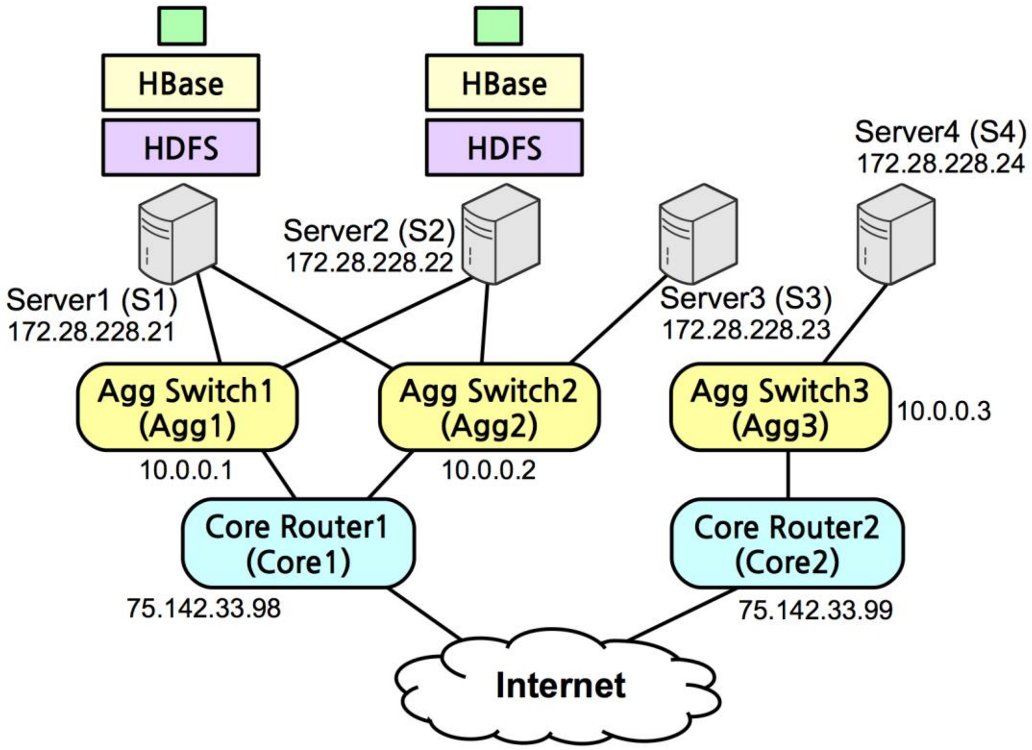
Repair



Specification:

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

Repair

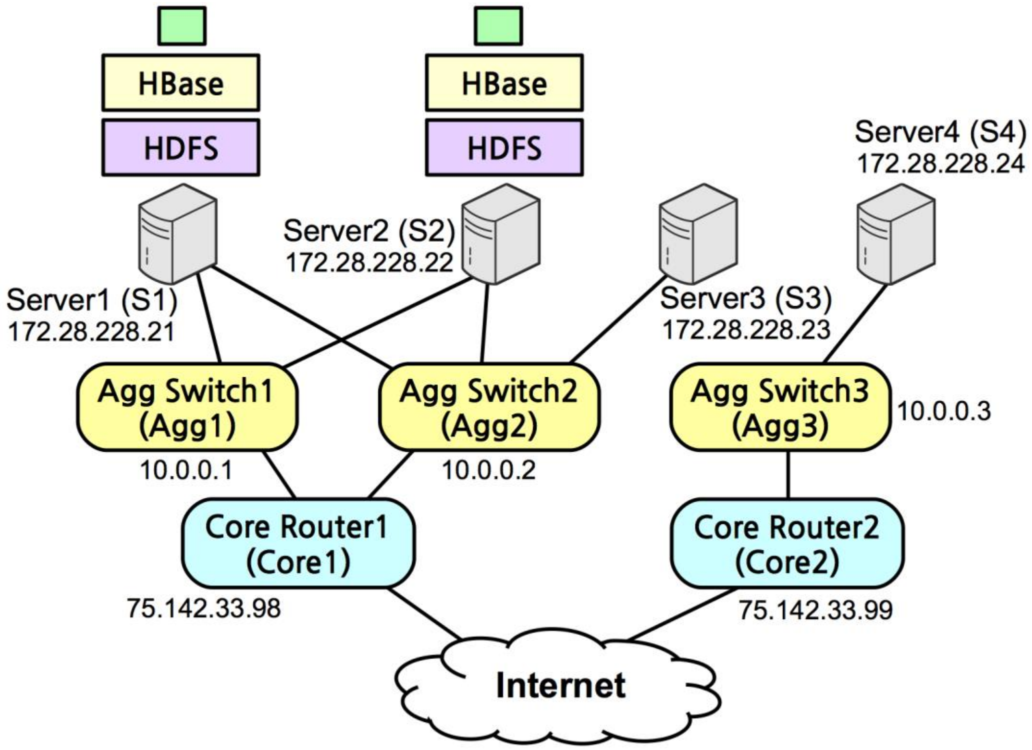


Specification:

$Server -> 172.28.228.21, 172.28.228.22

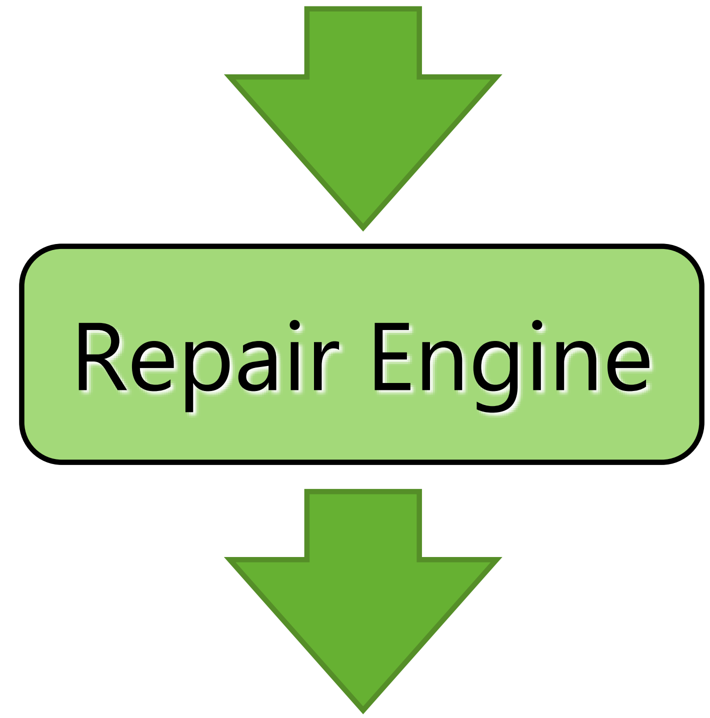
goal(failProb(ft)<0.08 | ChNode | Agg3)

Repair



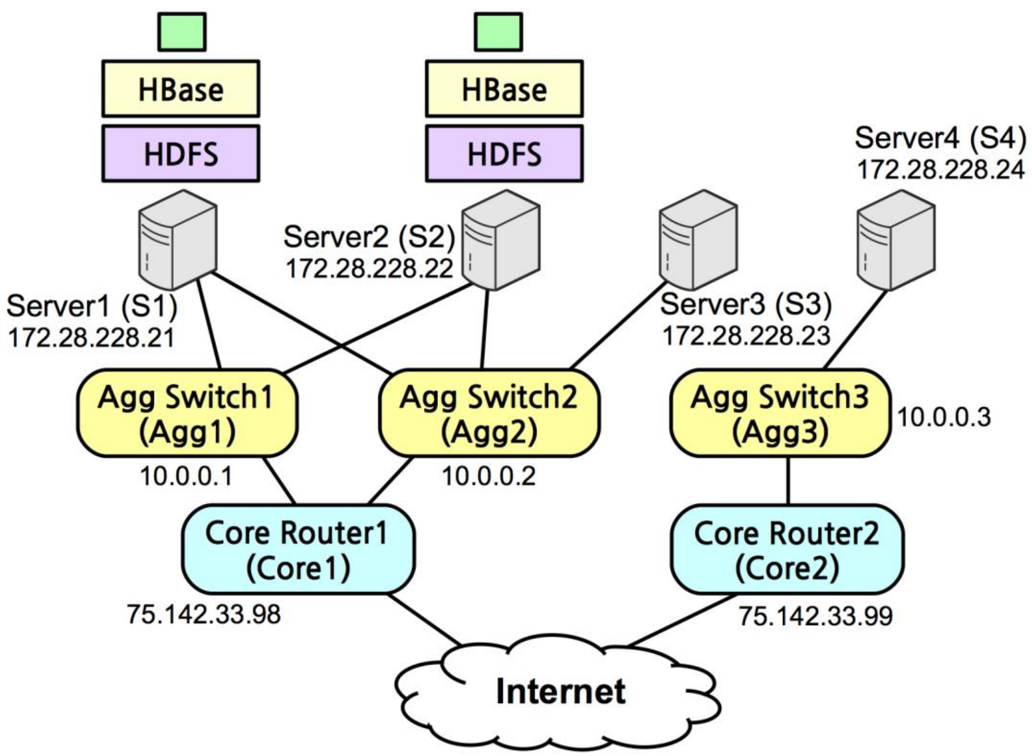
Specification:

$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

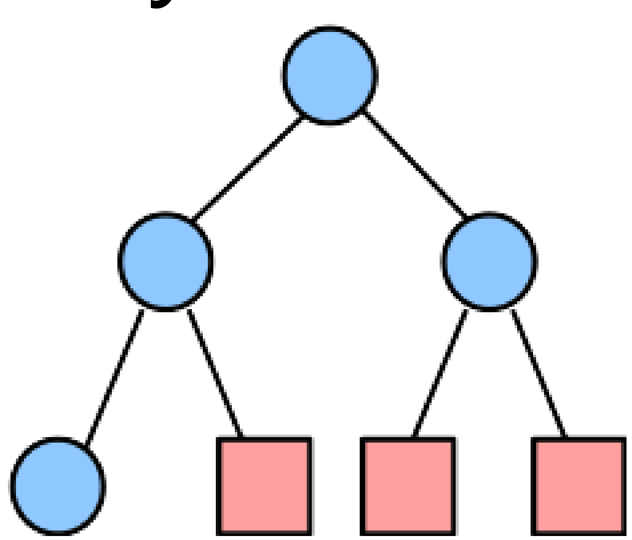


Specification:

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

Repair Engine

Synthesis



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

Expressiveness Evaluation

Auditing Tasks	RAL	Minimal cut set	Failure sampling
Modeling underlying topologies	4	213	224
Extracting and ranking RCGs	5	244	433
Computing failure probability	9	287	562
Ranking components	10	289	No support
Recommending the most independent deployments	16	562	1395

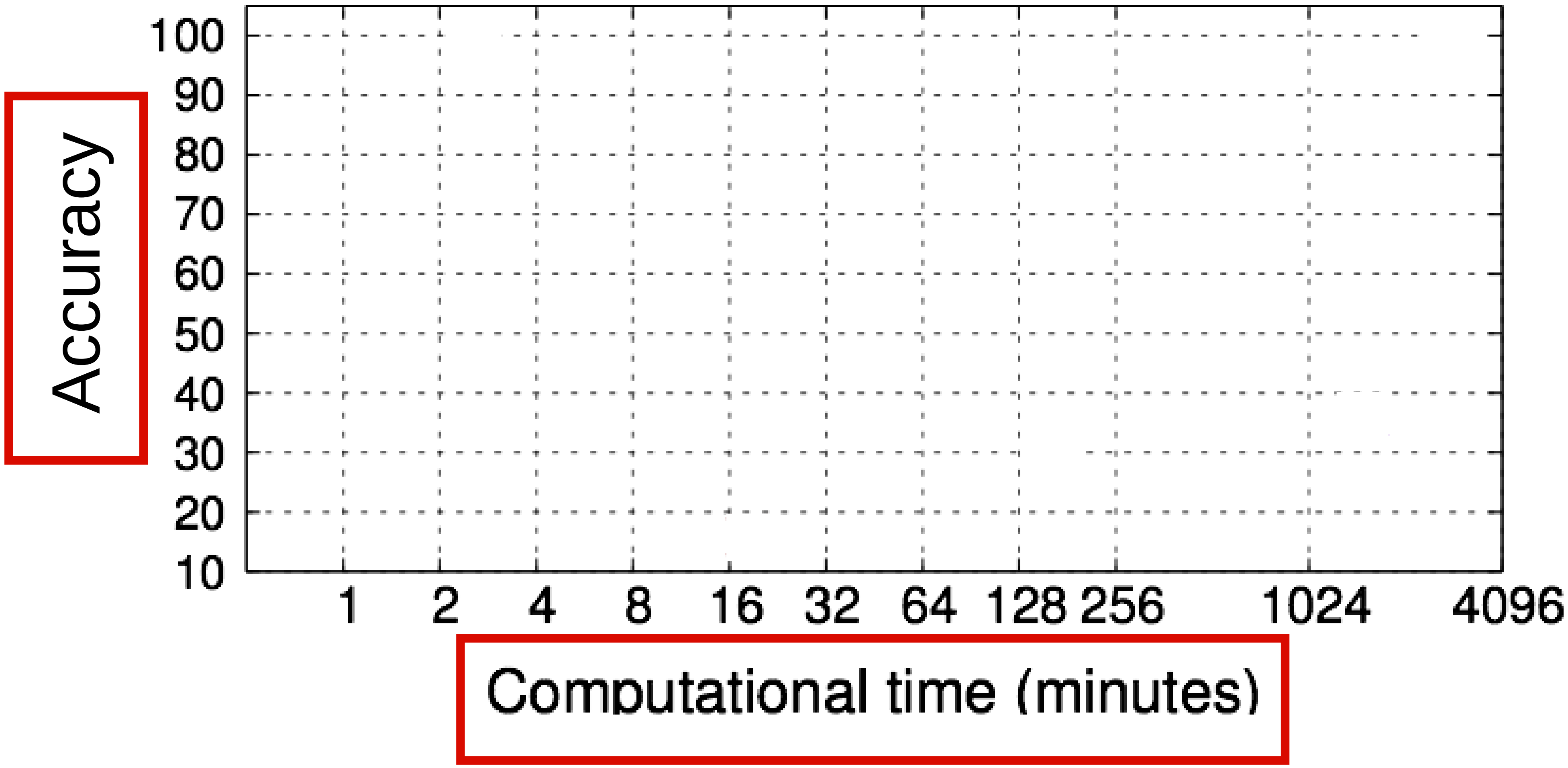
Fault Graph Analysis

	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

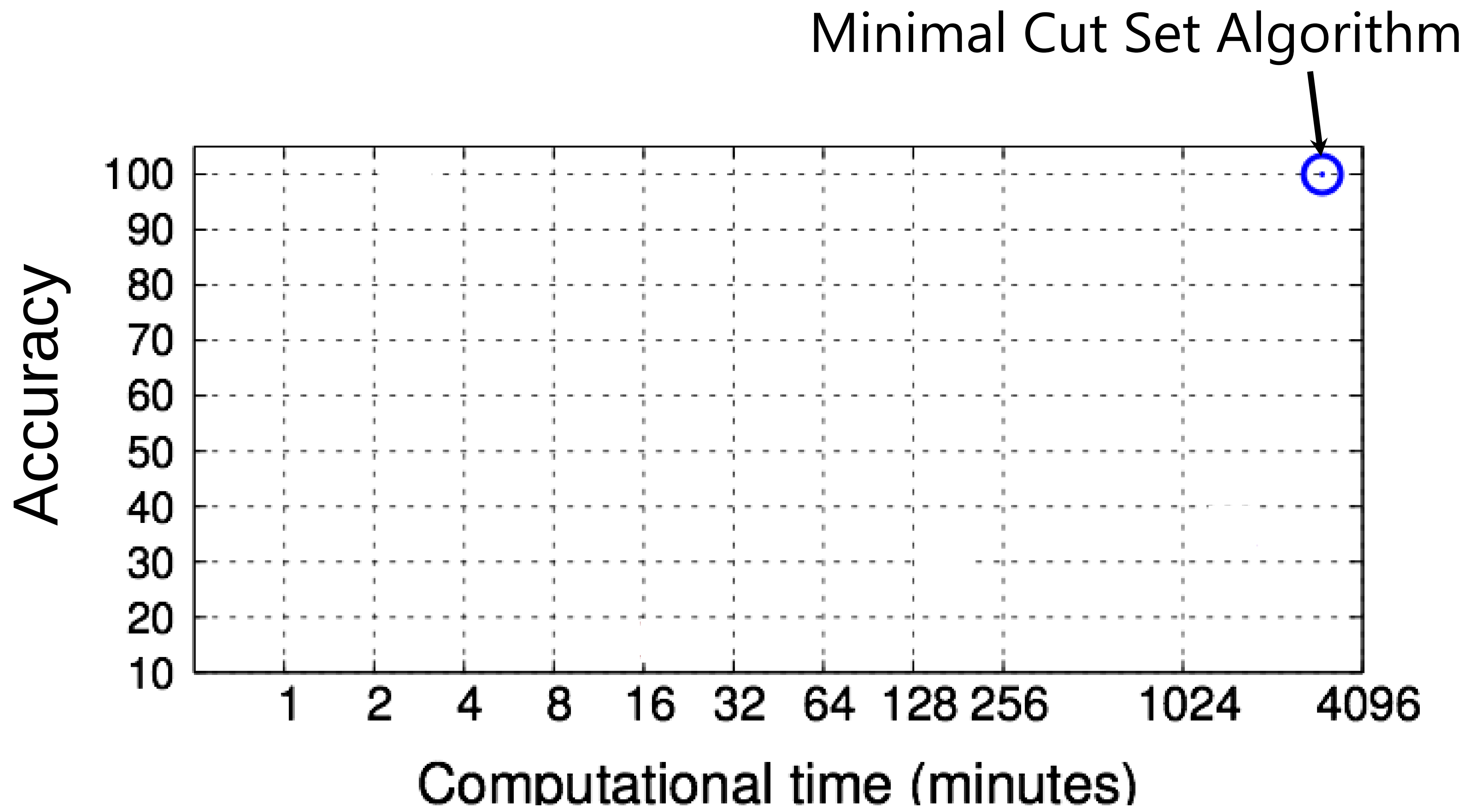
Fault Graph Analysis

	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

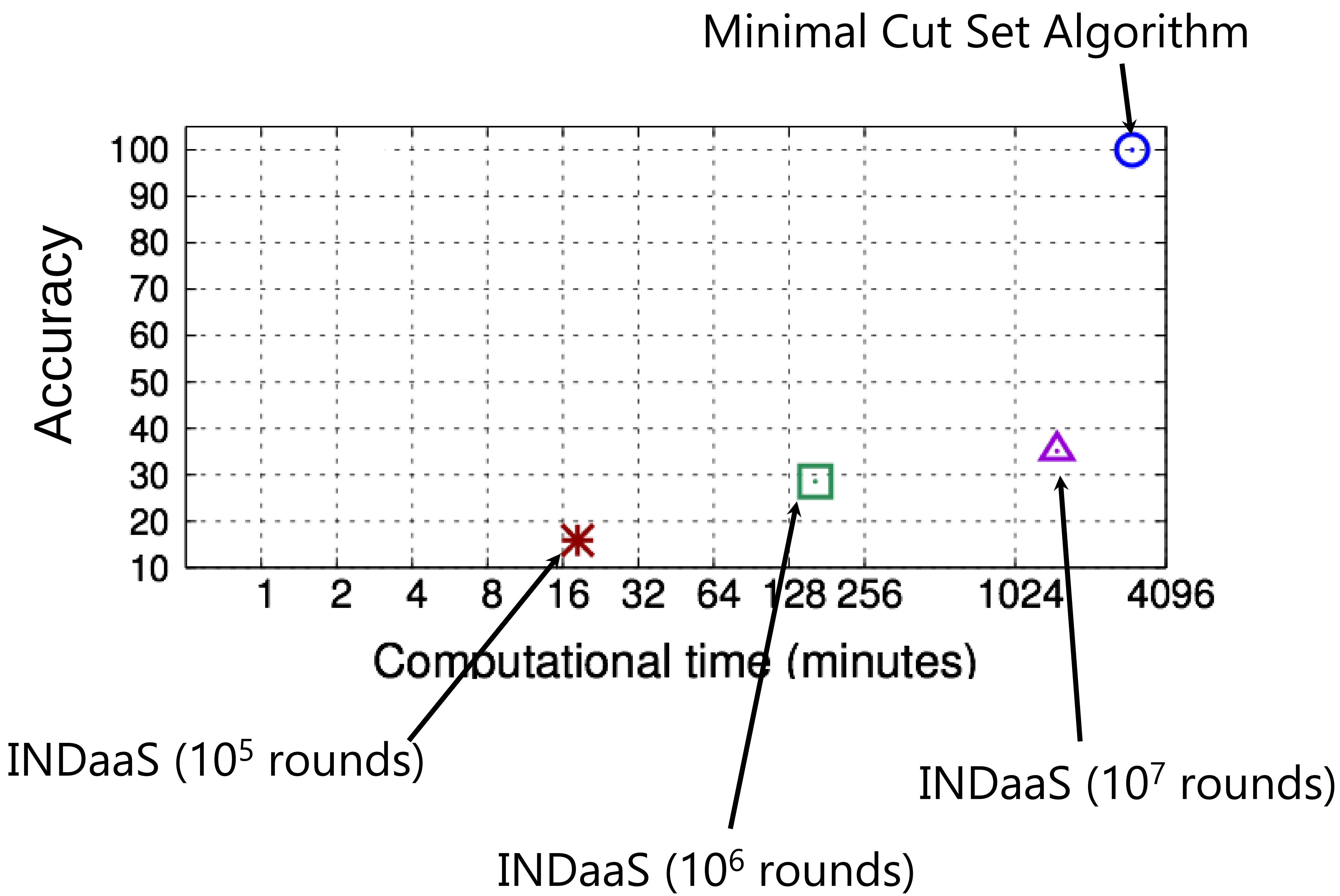
Topology C: 70,656 Nodes



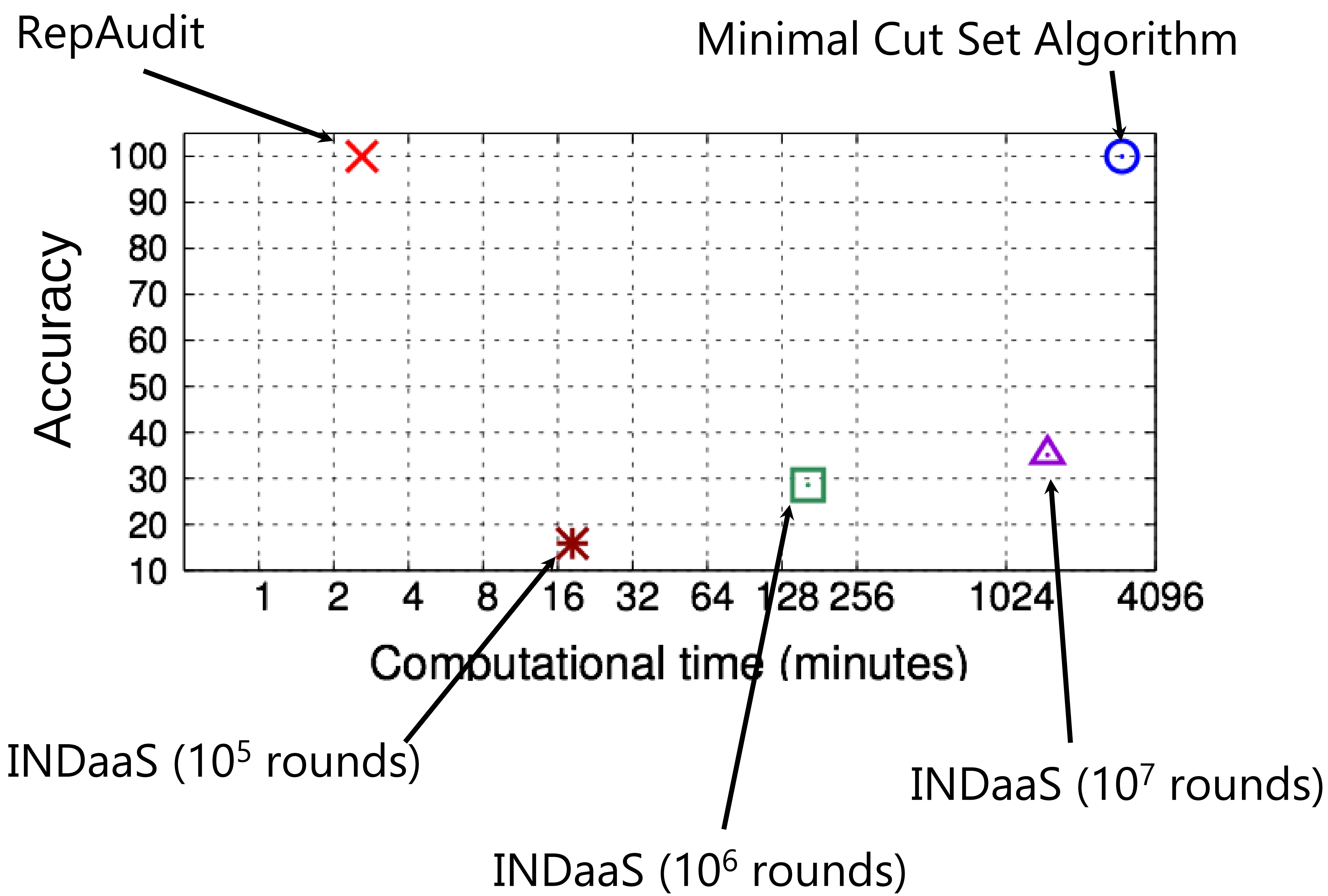
Topology C: 70,656 Nodes



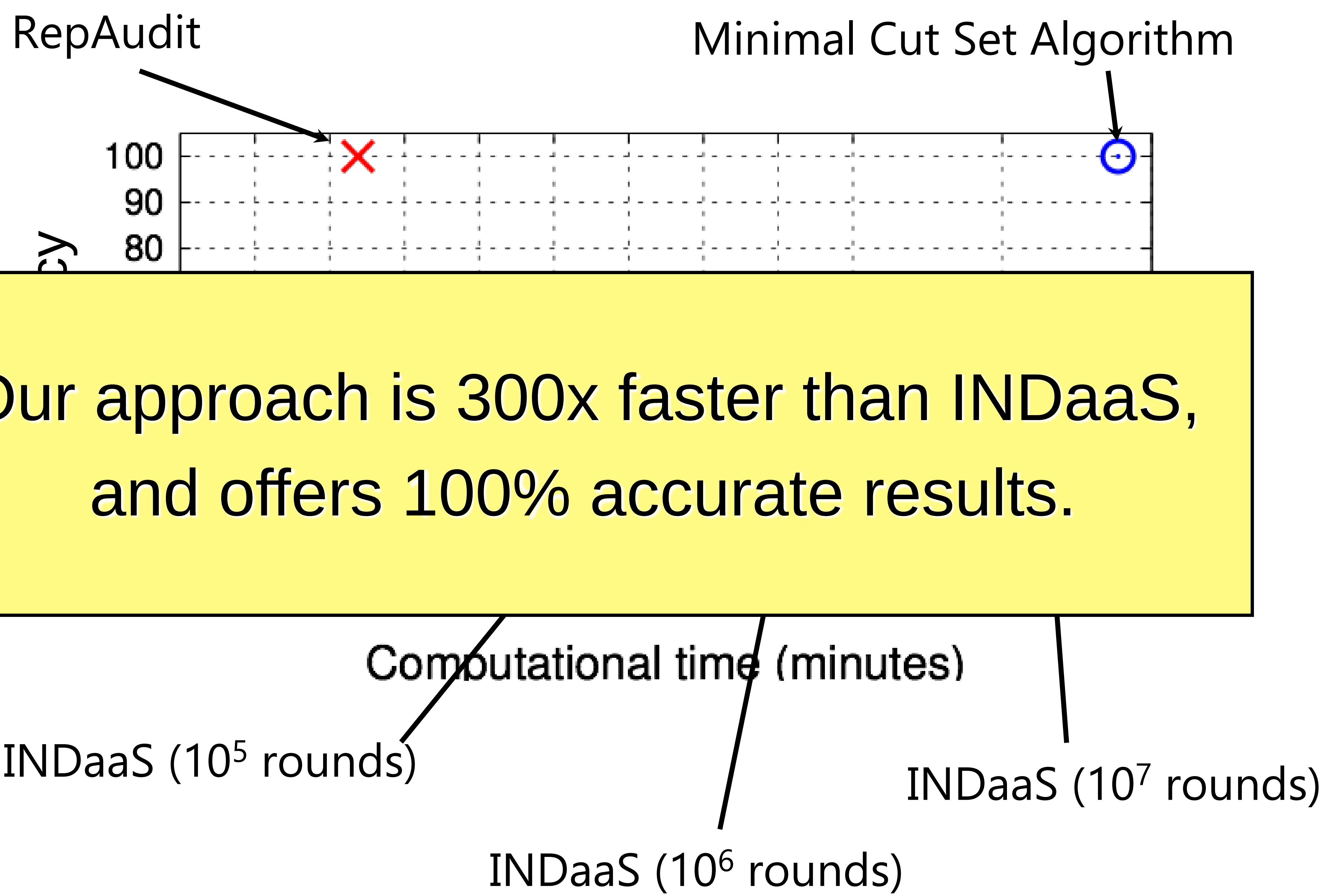
Topology C: 70,656 Nodes



Topology C: 70,656 Nodes



Topology C: 70,656 Nodes



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

Conclusion

- RepAudit is a language framework for auditing correlated failures in system runtime:
  - Flexible to express diverse auditing tasks
  - Accurate and rapid auditing capabilities
  - Useful to build new applications (e.g., repair)
- Source code publicly available at:
  - http://github.com/ennanzhai/repaudit