Heading off Correlated Failures through Independence-as-a-Service

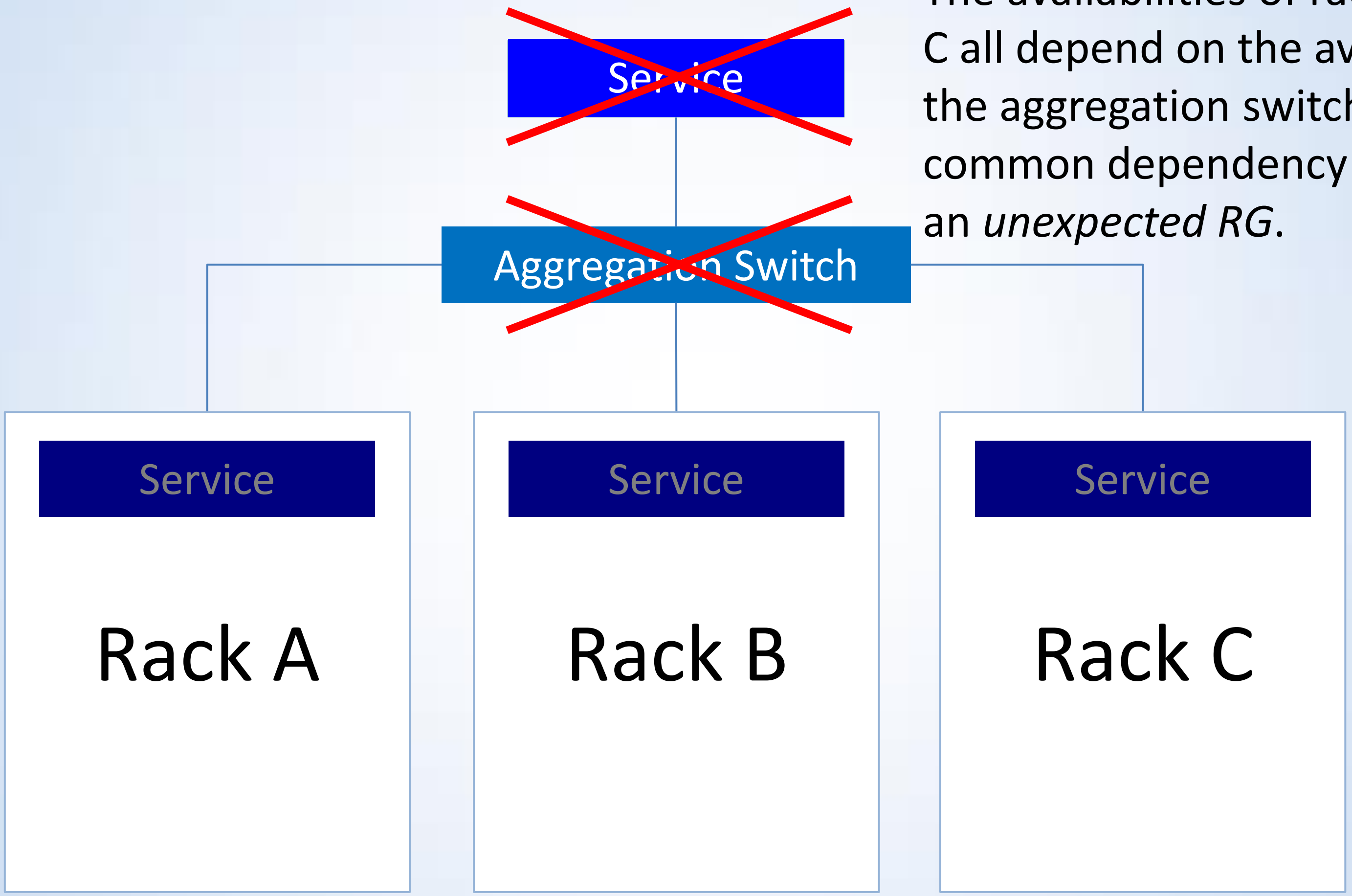
Ennan Zhai, Ruichuan Chen, David Isaac Wolinsky, and Bryan Ford

Presenter: Ron Wright

Motivation

- Cloud services depend on redundancy to ensure high reliability
- However, components that appear to be independent may share subtle dependencies, leading to unexpected *correlated failures*
- Redundant systems may contain *risk groups* (RGs), or sets of components that can cause a service outage if all the components fail simultaneously

What Can Go Wrong?



The availabilities of racks A, B, and C all depend on the availability of the aggregation switch. This common dependency introduced an *unexpected RG*.

Documented Examples

- Amazon AWS
  - One glitch on an EBS server disabled entire service across Amazon's US-East region
  - This, in turn, caused correlated failures among EC2 instances utilizing the EBS server, which disabled applications designed for EC2 redundancy
- Google Storage
  - "Close to 37% of failures are truly correlated"
  - No tools to identify failure correlations systematically
- iCloud
  - A storm in Dublin disabled both Amazon and Microsoft clouds in that region for hours

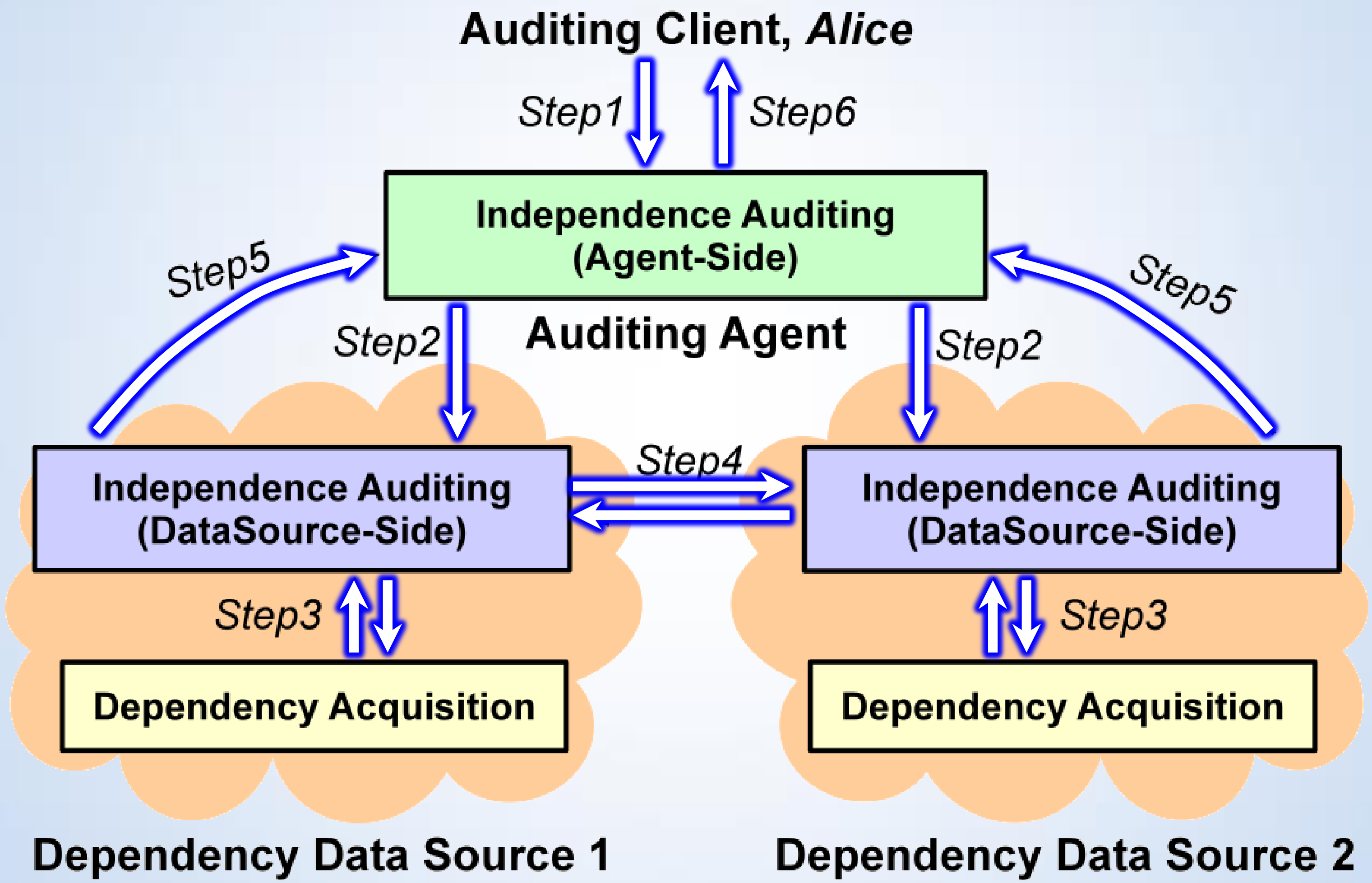
Independence-as-a-Service (INDaaS)

- Architecture that proactively collects and audits structural dependency data to evaluate independence of redundant systems before any failures occur
  - *Dependency acquisition modules* collect dependency data
  - *Auditing modules* quantify independence of redundant systems and pinpoint common dependencies that may cause correlated failures

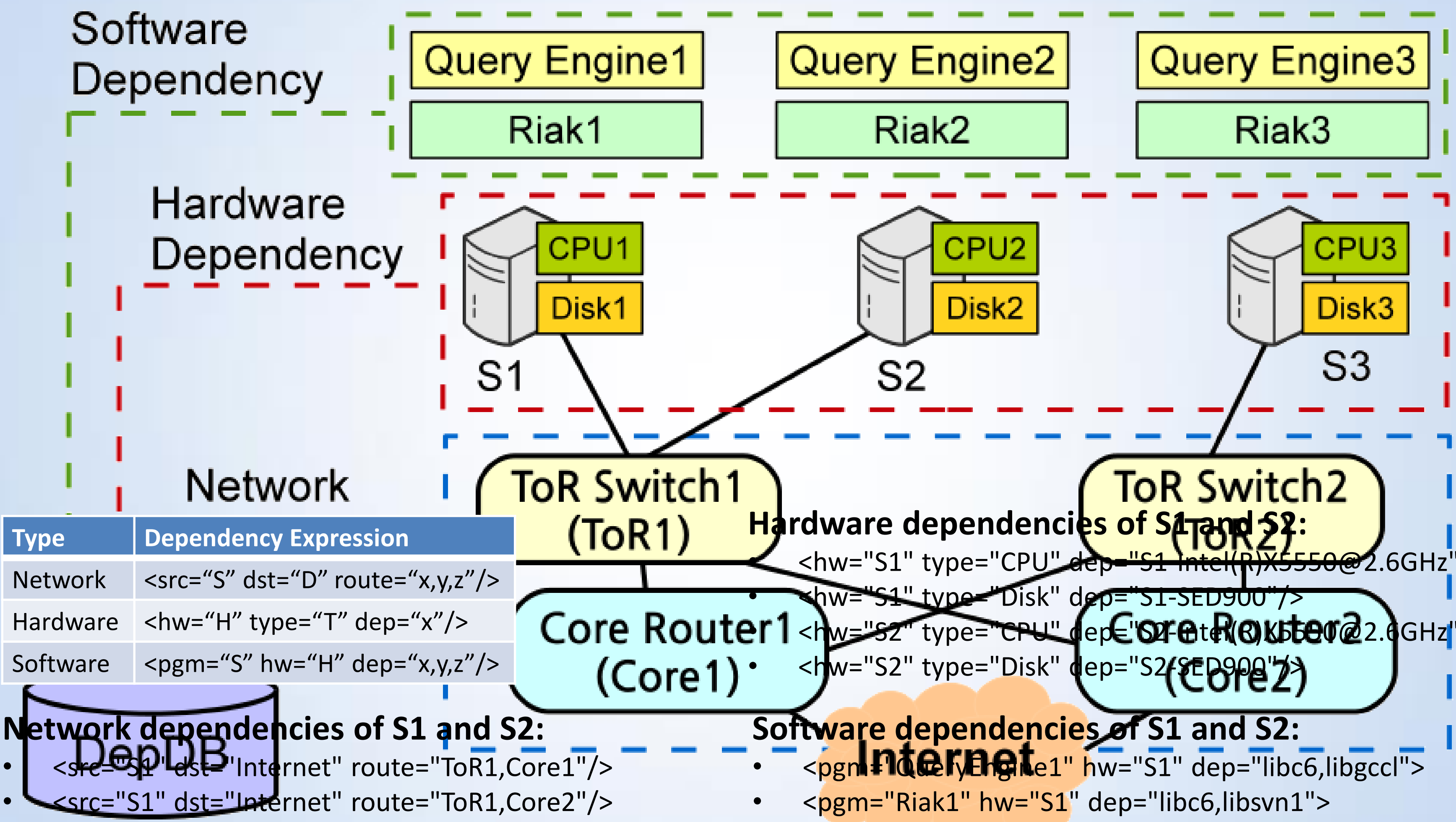
Main Contributions

1. Evaluates independence of redundant systems before or during deployment
2. Provides fault graph analysis to enable the evaluation of dependencies at multiple levels of detail
3. Uses scalable fault graph analysis
4. Supports efficient PIA through private set intersection cardinality
5. Provides realistic case studies with a prototype implementation

Architecture



Dependency Data Representation



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”/>

Hardware dependencies of S1 and S2:

- <hw="S1" type="CPU" dep="S1-Intel(R)X5550@2.6GHz"/>
- <hw="S1" type="Disk" dep="S1-SED900"/>
- <hw="S2" type="CPU" dep="S2-Intel(R)X5550@2.6GHz"/>
- <hw="S2" type="Disk" dep="S2-SED900"/>

Network dependencies of S1 and S2:

- <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"/>

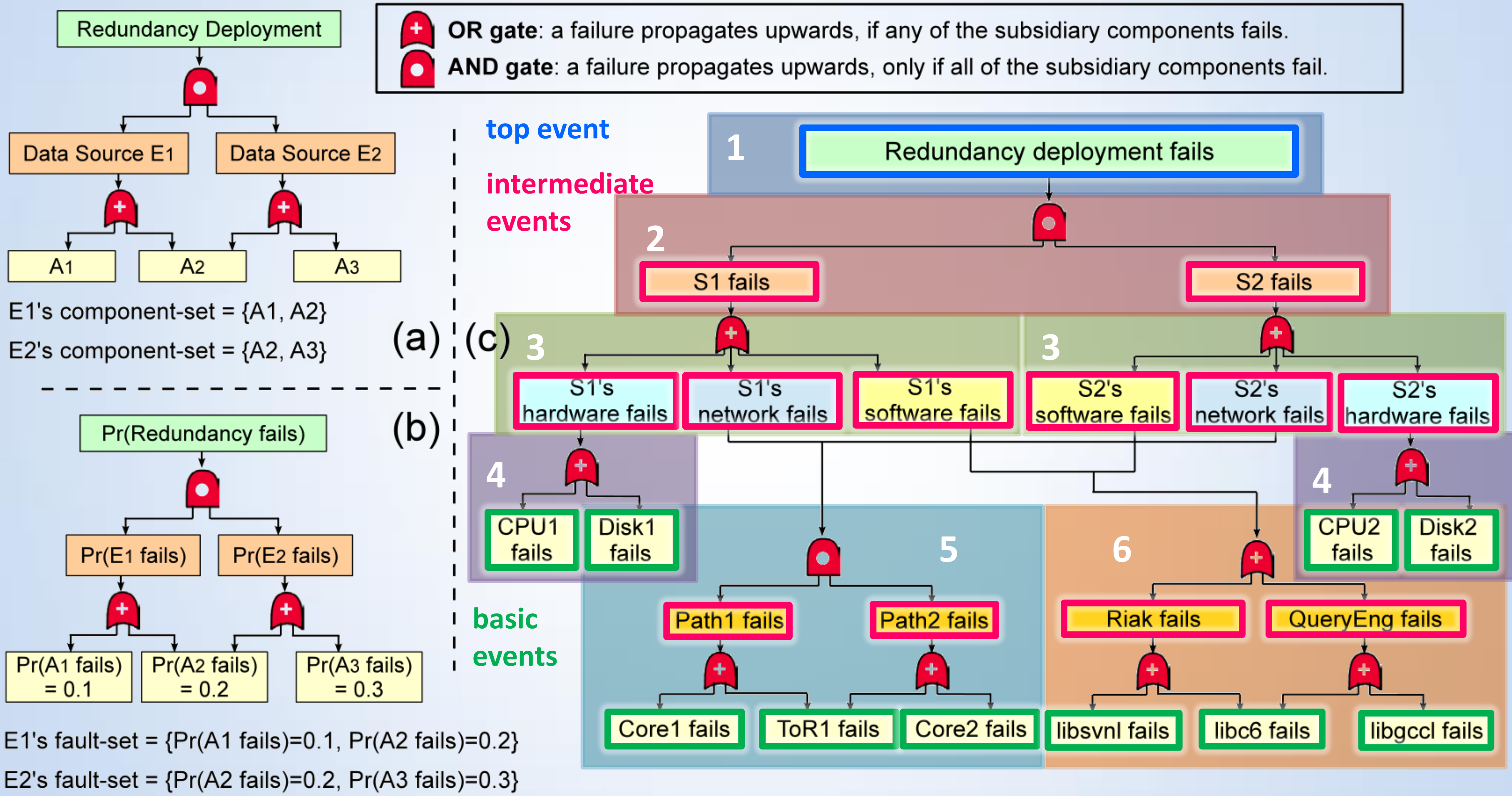
Software dependencies of S1 and S2:

- <pgm="QueryEngine1" hw="S1" dep="libc6,libgccl">
- <pgm="Riak1" hw="S1" dep="libc6,libsvn1">
- <pgm="QueryEngine2" hw="S2" dep="libc6,libgccl">
- <pgm="Riak2" hw="S2" dep="libc6,libsvn1">

Structural Independence Auditing (SIA)

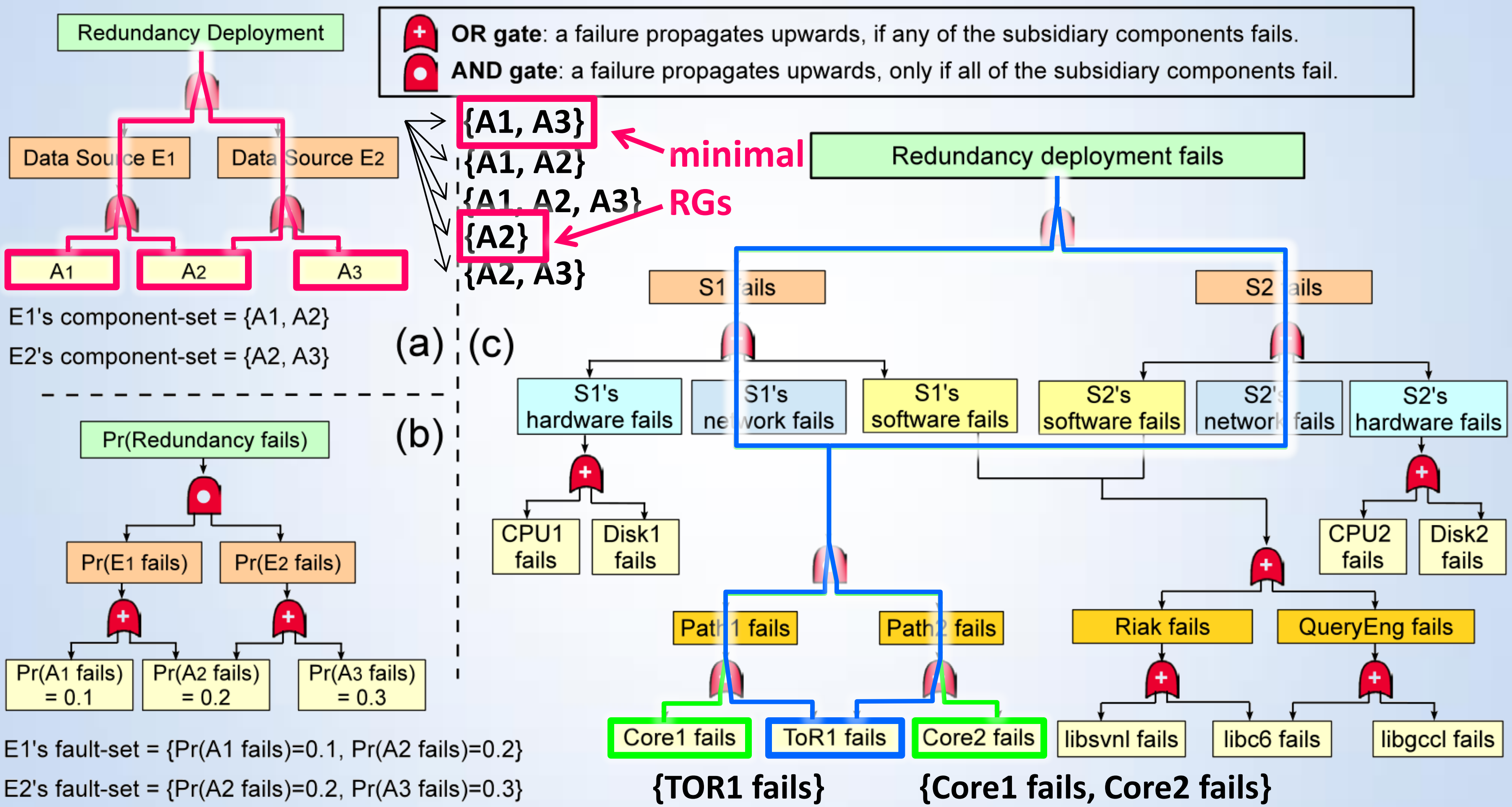
- Assumes data sources are willing to share full dependency data with each other
- Involves generating a dependency graph, finding and ranking risk groups, and generating an audit report

Dependency Graph



(a) Component-set **(b) Fault-set** **(c) Fault graph**

Risk Groups in Dependency Graphs



(a) Component-set **(b) Fault-set** **(c) Fault graph**

Algorithms for Finding Risk Groups

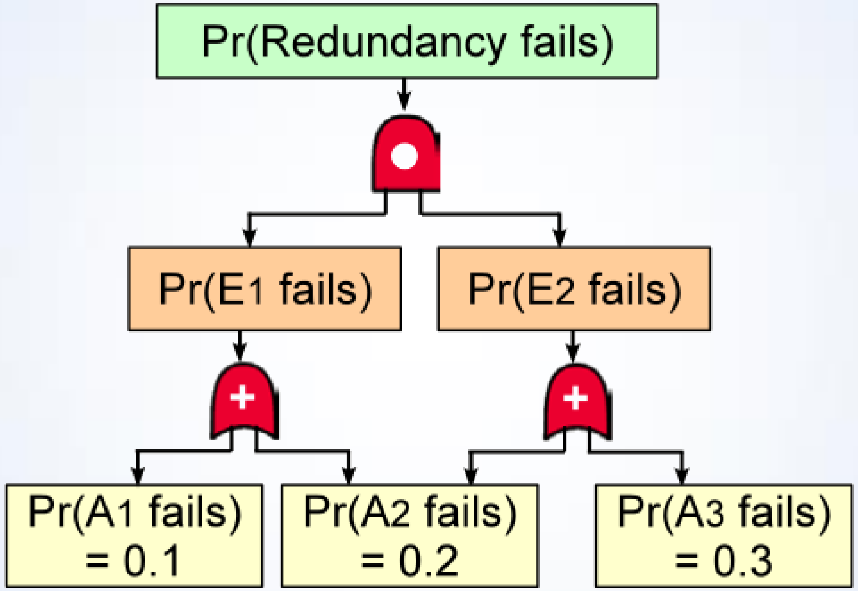
- Minimal RG algorithm
  - Directly computes minimal RGs using reverse breadth-first traversal
  - Pros
    - Results are exact
  - Cons
    - Algorithm is NP hard!
- Failure sampling algorithm
  - Randomly assigns 0s and 1s to basic events to test for deployment failure and generate the appropriate RGs
  - Pros
    - Linear time complexity
  - Cons
    - Non-deterministic
    - No guarantee that any RG is minimal

Ranking Risk Groups

- Size-based ranking
  - Ranks RGs based on the number of components in each RG
  - The smaller the number of components in the RG, the higher the rank
- Failure probability ranking
  - Ranks RGs based on their relative importance, $I_c = Pr(C) / Pr(T)$
    - Pr(C) represents probability of any given failure event C
    - Pr(T) represents probability of any given failure event T
  - Pr(T) computed by inclusion-exclusion principle involving all minimal RGs of T

Failure Probability Ranking Example

Pr(T) = Pr(A1 and A3 fail, or A2 fails) = 0.1 · 0.3 + 0.2 – 0.1 · 0.3 · 0.2 = 0.224



- $\mathbf{I}_{\textbf{A2 fails}}$ = Pr(A2 fails) / Pr(T) = 0.2 / 0.224 = **0.8929**
- $\mathbf{I}_{\textbf{A1 fails, A3 fails}}$ = Pr(A1 fails, A3 fails) / Pr(T) = 0.1 · 0.3 / 0.224 = **0.1339**

Therefore, the RG {A2 fails} is ranked higher than the RG {A1 fails, A3 fails}.

Generating the Audit Report

- Let R denote a specific redundancy deployment
- Let c_i denote the i-th RG in R's RG-ranking list
- Size-based ranking algorithm
  - $indep(R) = \sum_{i=1}^{n} size(c_i)$
- Failure probability ranking algorithm
  - $indep(R) = \sum_{i=1}^{n} I_{c_i}$
- Computed independence scores, returned to the client, can be used to choose the most independent deployment for a particular service, for example

Private Independence Auditing (PIA)

- Allows auditing to take place, even across two *cloud providers* unwilling to share full dependency data with each other
- Trust assumptions
  1. Auditing clients may be malicious and would like to know as much as possible about the providers' dependency data
  2. Cloud providers and auditing agents are honest but curious
  3. No collusion among cloud providers and auditing agents

Jaccard similarity

- Let S_i denote the i-th dataset
- $J(S_0, \cdots, S_{k-1}) = \frac{|S_0 \cap \cdots \cap S_{k-1}|}{|S_0 \cup \cdots \cup S_{k-1}|}$
- Above computation useful for small datasets
- Low similarity for J close to 0, high similarity for J close to 1, significant correlation for J greater than or equal to 0.75

MinHash

- An approximation to Jaccard similarity, which is useful for large datasets
- Let $h^{(1)}(\cdot)$, …, $h^{(m)}(\cdot)$ denote m different hash functions
- MinHash constructs a vector $\left\{h_{min}^{(i)}(S)\right\}_{i=1}^{m}$ and computes Jaccard similarity as $J(S_0, \cdots, S_{k-1}) = \frac{\delta}{m} + O\left(\frac{1}{\sqrt{m}}\right)$, where
  - δ denotes the number of datasets satisfying
$$h_{min}^{(i)}(S_1) = \cdots = h_{min}^{(i)}(S_{k-1})$$
  - $h_{min}^{(i)}(S)$ denotes an item $e \in S$ with the smallest value $h^{(1)}(e)$

P-SOP

<u>Key assumptions:</u>

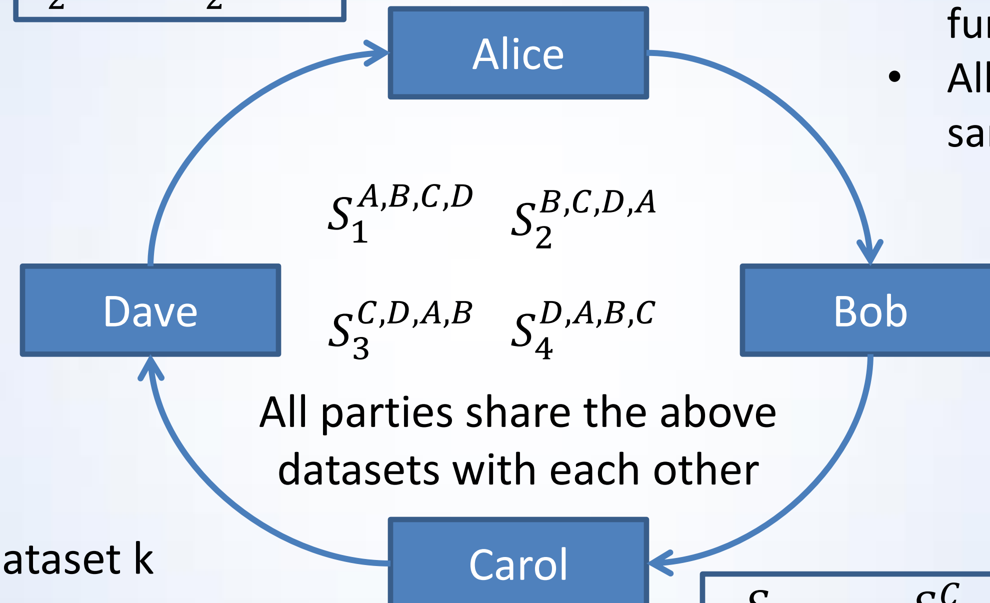
- Each party encrypts the datasets using commutative encryption
- Each party permutes its dataset elements using a fixed permutation function
- All parties agree on the same hash function

$S_1 \quad S_1^A$

$S_4^D \quad S_4^{D,A}$

$S_3^{C,D} \quad S_3^{C,D,A}$

$S_2^{B,C,D} \quad S_2^{B,C,D,A}$



$S_1^{A,B,C,D} \quad S_2^{B,C,D,A}$

$S_3^{C,D,A,B} \quad S_4^{D,A,B,C}$

All parties share the above datasets with each other

$S_2 \quad S_2^B$

$S_1^A \quad S_1^{A,B}$

$S_4^{D,A} \quad S_4^{D,A,B}$

$S_3^{C,D,A} \quad S_3^{C,D,A,B}$

$S_3 \quad S_3^C$

$S_2^B \quad S_2^{B,C}$

$S_1^{A,B} \quad S_1^{A,B,C}$

$S_4^{D,A,B} \quad S_4^{D,A,B,C}$

$S_4 \quad S_4^D$

$S_3^C \quad S_3^{C,D}$

$S_2^{B,C} \quad S_2^{B,C,D}$

$S_1^{A,B,C} \quad S_1^{A,B,C,D}$

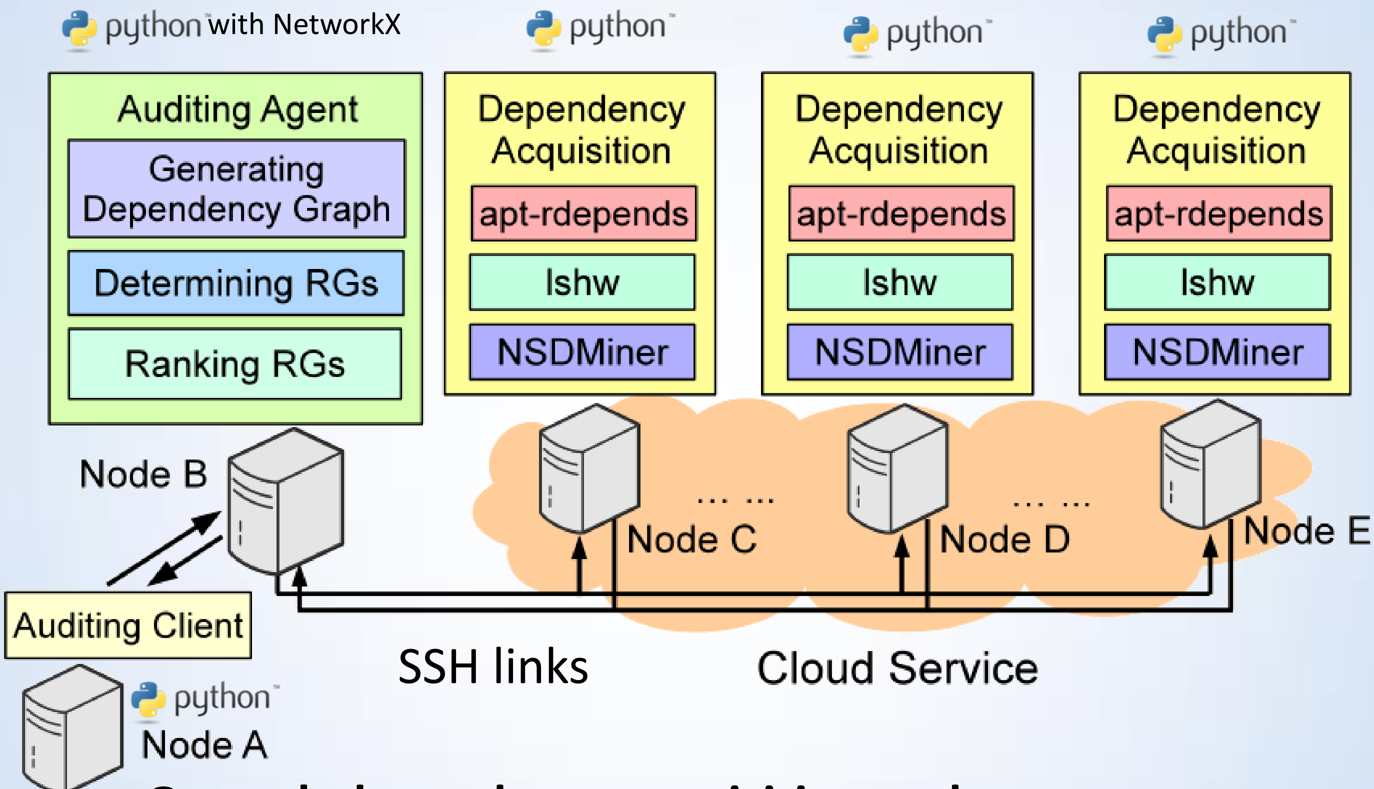
S_k = the original dataset k

$S_k^{p_1,p_2,p_3,\ldots}$ = dataset k hashed, encrypted, and permuted by p_1; then encrypted and permuted by p_2; then encrypted and permuted by p_3; etc.

Dependency Graph & Audit Report

- Each provider first generates local dependency graph at component-set level
- Each provider *normalizes* generated component-set S_i using two types of components with common correlated failures
  - Third-party routing elements (e.g., ISP routers)
    - Accessible IP addresses used as unique identifiers
  - Third-party software packages (e.g., OpenSSL)
    - Standard names plus software versions used as unique identifiers
- Report consists of rankings of Jaccard similarities

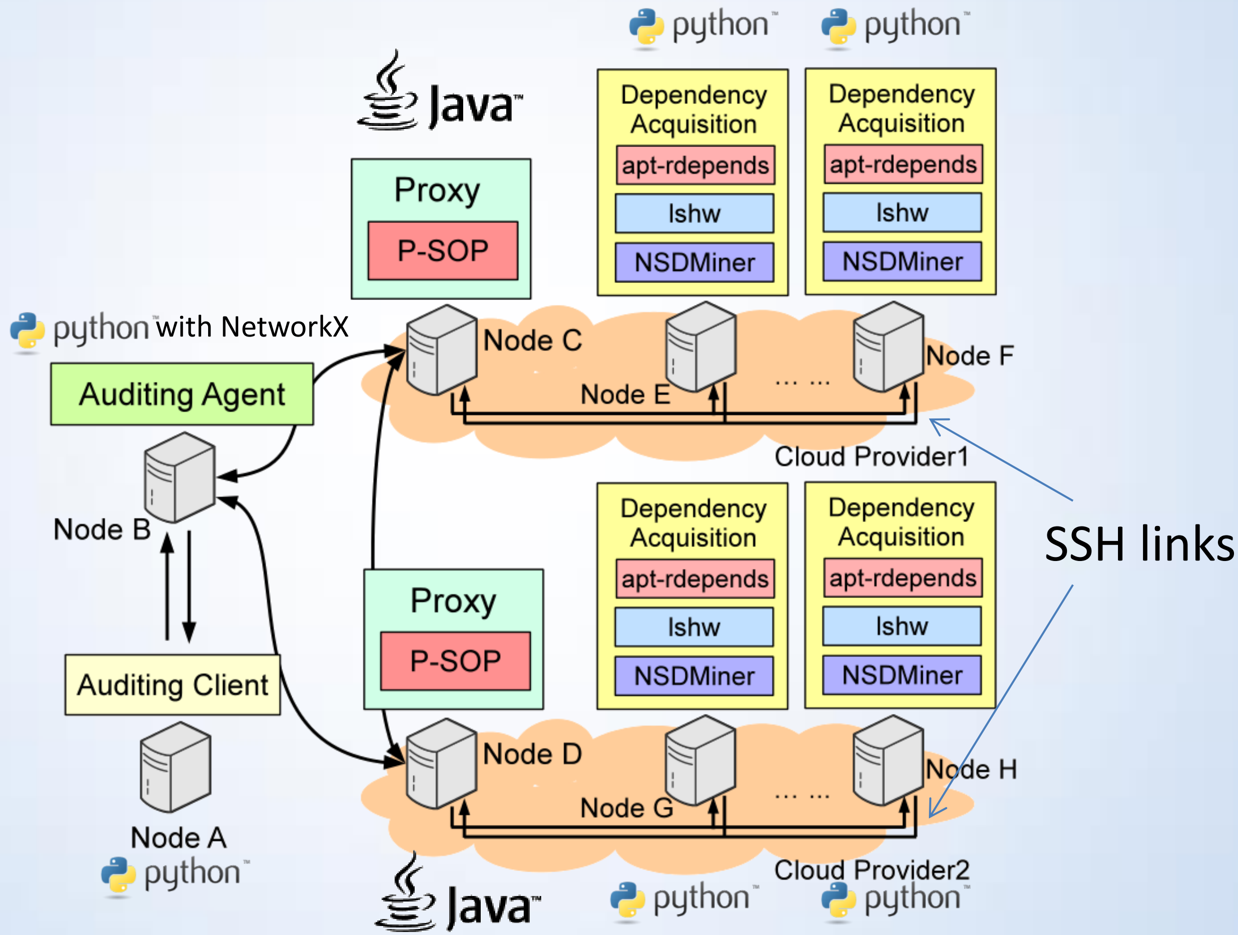
SIA Implementation & Deployment



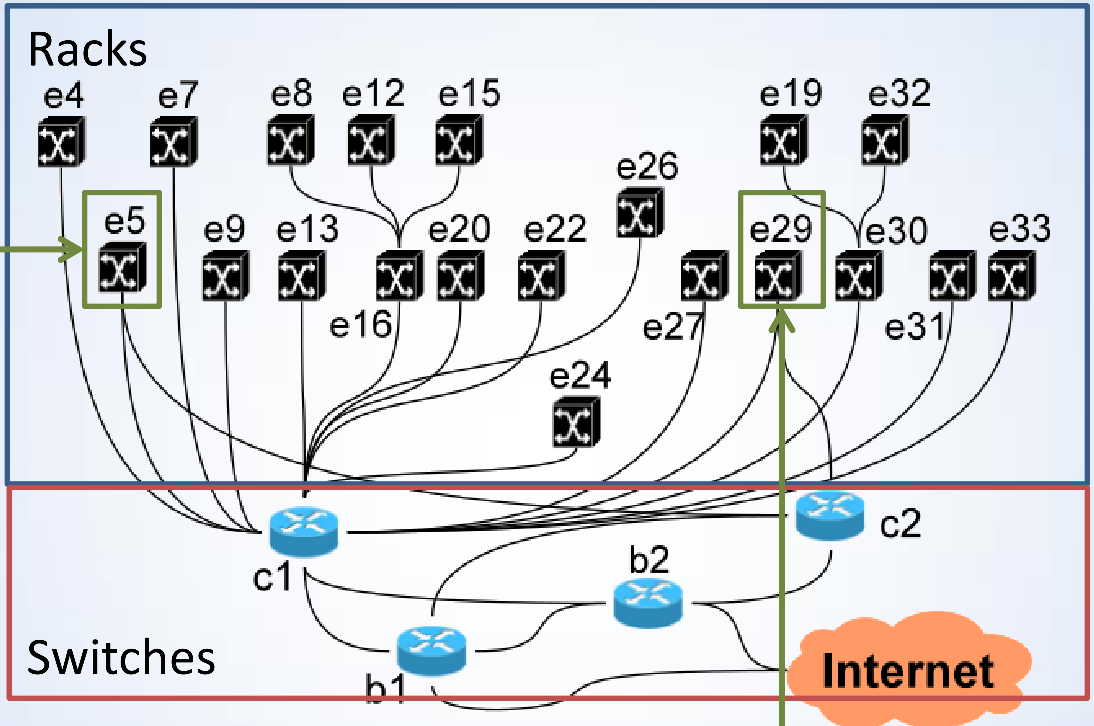
On each dependency acquisition node:

- NSDMiner used for network dependencies
- lshw used for hardware dependencies
- apt-rdepends used for software dependencies

PIA Implementation & Deployment



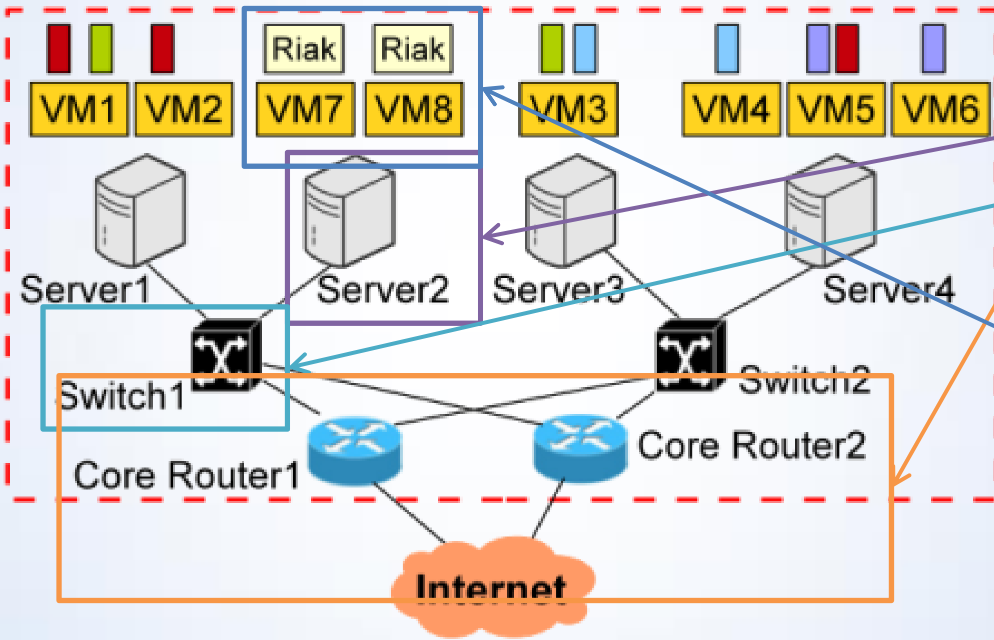
Network Dependency Case Study



(a) Common network dependency.

{Rack 5, Rack 29} most independent deployment

Hardware Dependency Case Study

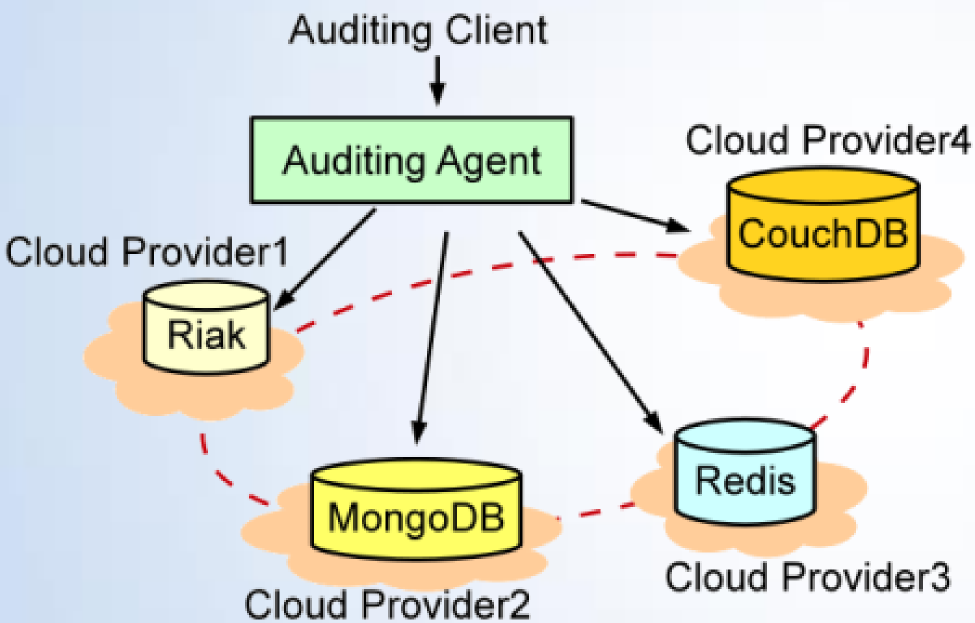


(b) Common hardware dependency.

<u>Top-ranking RGs:</u>

1. {Server2}
2. {Switch1}
3. {Core1, Core2}
4. {VM7, VM8}

Software Dependency Case Study



(c) Common software dependency.

Rank	2-Way Redundancy	Jaccard
1	Cloud2 & Cloud4	0.1419
2	Cloud2 & Cloud3	0.1547
3	Cloud1 & Cloud4	0.2081
4	Cloud1 & Cloud3	0.2939
5	Cloud3 & Cloud4	0.3489
6	Cloud1 & Cloud2	0.5059

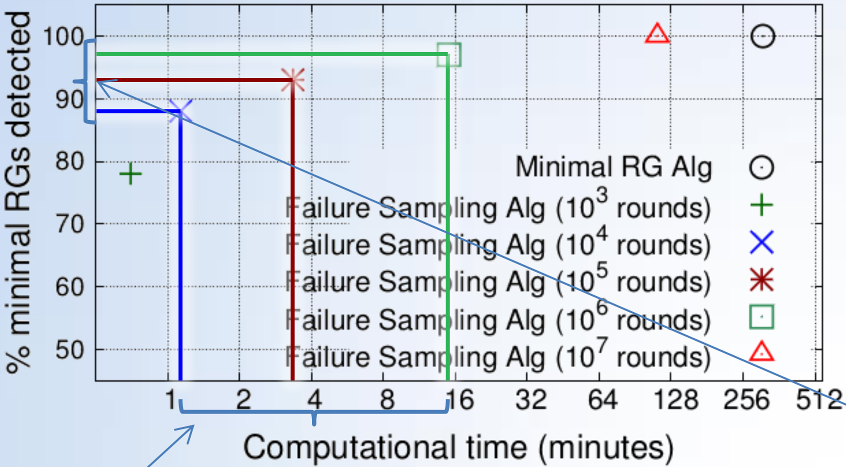
Rank	3-Way Redundancy	Jaccard
1	Cloud2 & Cloud3 & Cloud4	0.1128
2	Cloud1 & Cloud2 & Cloud4	0.1207
3	Cloud1 & Cloud3 & Cloud4	0.1353
4	Cloud1 & Cloud2 & Cloud3	0.1536

Performance Evaluation Configuration

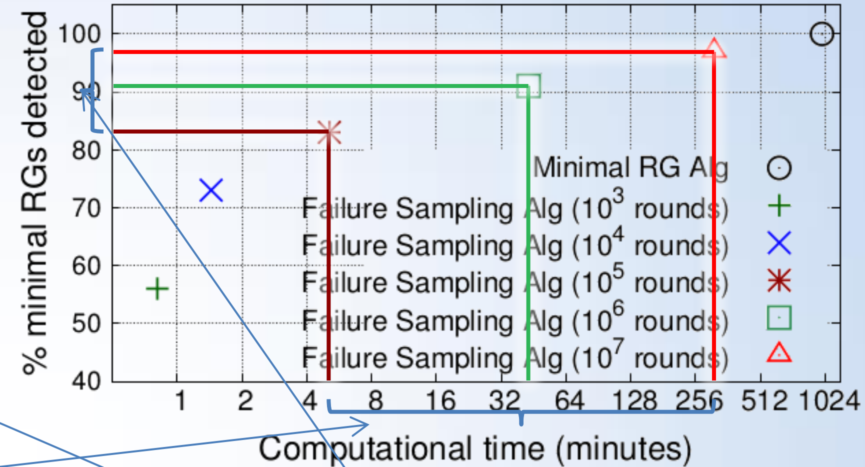
Performance of INDaaS was evaluated on 40 workstations containing Intel Xeon Quad Core HT 3.7 GHz CPUs and 16 GB of RAM.

	Topology A	Topology B	Topology C
# switch ports	16	24	48
# core routers	64	144	576
# agg switches	128	288	1,152
# ToR switches	128	288	1,152
# servers	1,024	3,456	27,648
Total # of devices	1,344	4,176	30,528

SIA Performance Evaluation Results

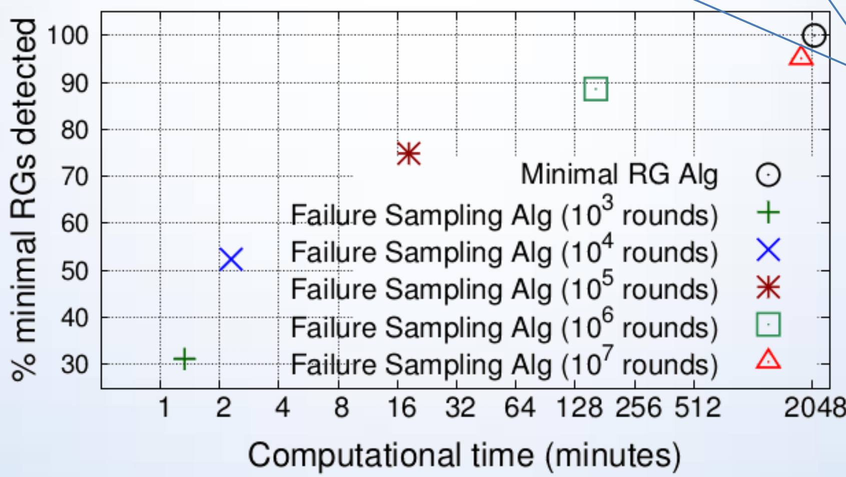


(a) Topology A: 1, 344 devices.



(b) Topology B: 4, 176 devices.

(Roughly) linear computation performance seen for failure sampling algorithm in all topologies



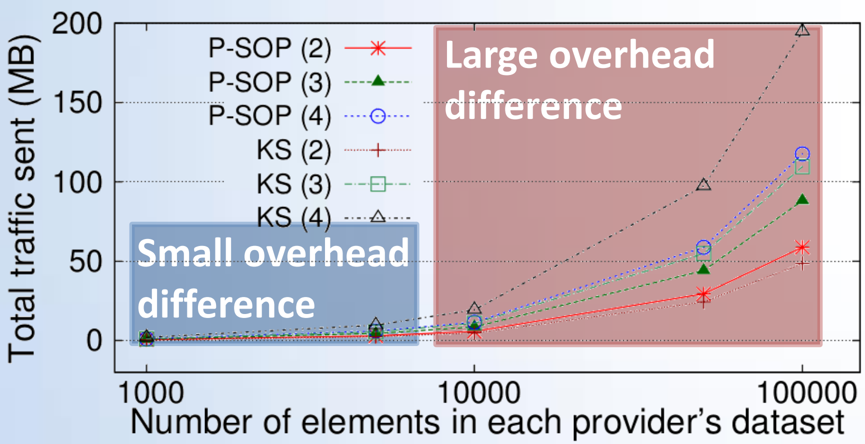
(c) Topology C: 30, 528 devices.

Tradeoff exists between linear time complexity and logarithmic percentage of minimal RGs detected

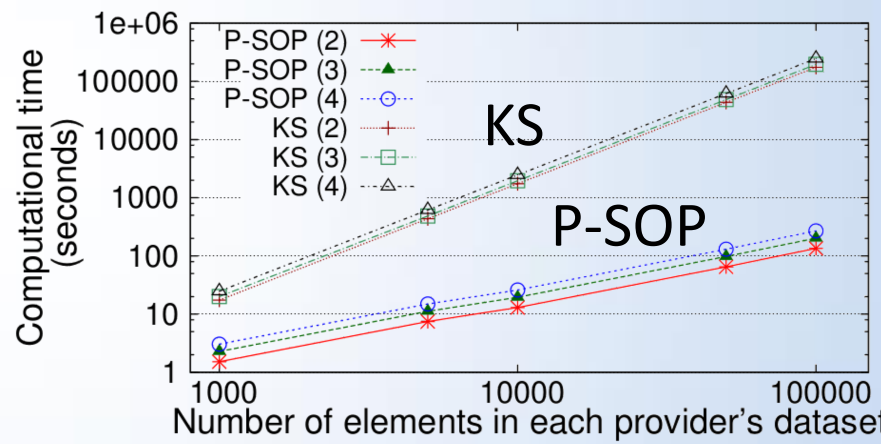
PIA Performance Evaluation Results

- 1024-bit keys were used for all types of encryption

P-SOP consistently outperforms KS computation-wise.

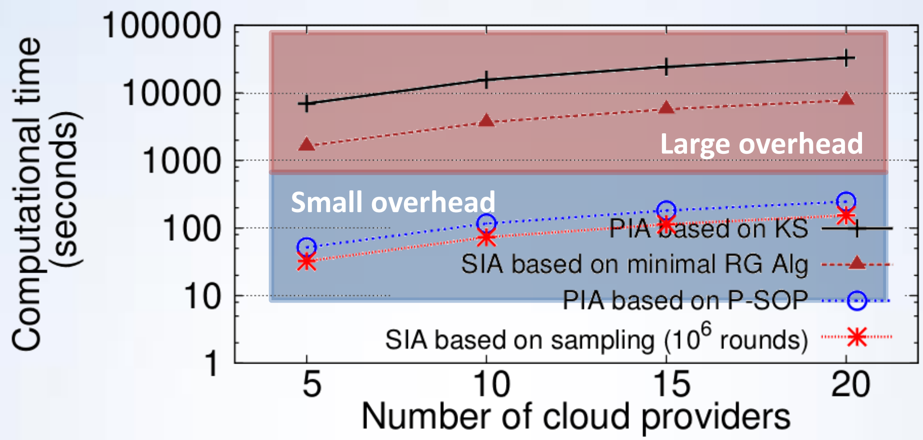


(a) Bandwidth overhead.

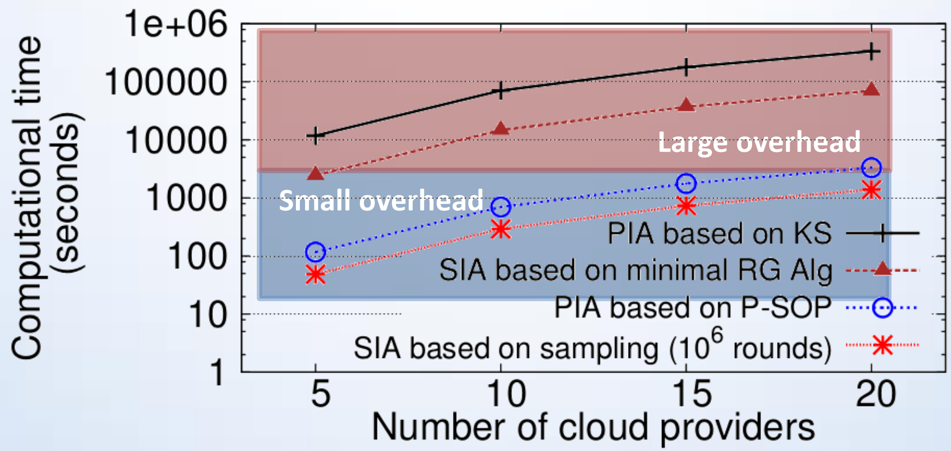


(b) Computational overhead.

Comparing Performance of SIA & PIA



(a) Two-way redundancy.



(b) Three-way redundancy.

In both cases

- **P-SOP outperforms KS**
- **10^6 rounds of random sampling outperform minimal RG algorithm**
- **Minimal RG algorithm and KS do not scale well**

Comments & Criticisms

- Pros
  - Risk group ranking makes it easy for users to identify potential correlated failures in deployment configurations
  - Flexible in allowing cloud providers to decide whether to share their dependency data with other cloud providers
- Cons
  - For large enough deployments, in some cases, failure sampling algorithm may run longer with much fewer minimal RGs than the minimal RG algorithm
  - Cannot be used for complex dependency acquisition
  - Trust assumptions may not hold in reality (e.g., cloud providers may behave maliciously)
  - Cannot have fault-set level dependency graphs and failure probability-based ranking without accurate failure probability information
  - INDaaS is not fault tolerant in itself (e.g., the P-SOP nodes in PIA and the auditing agent are single points of failure)

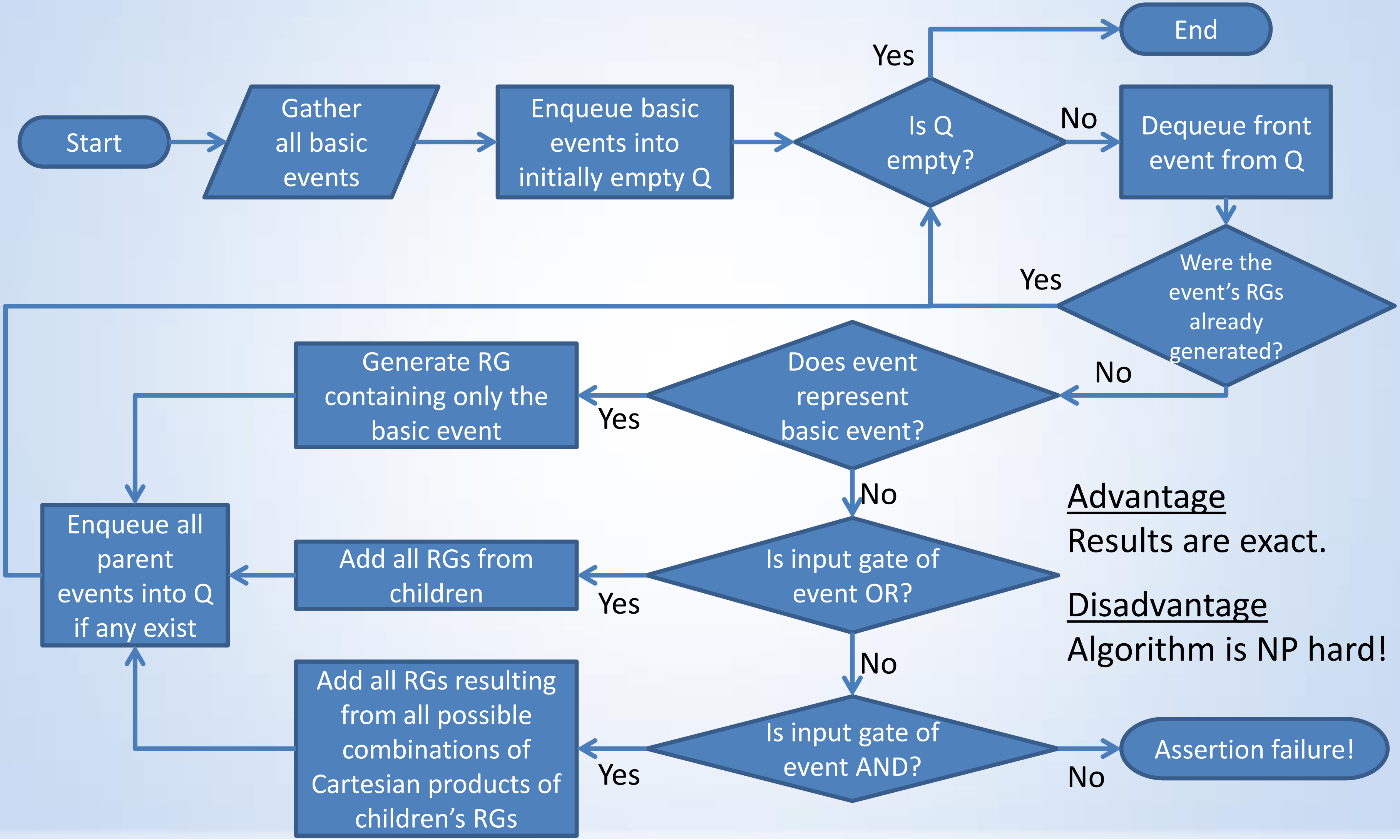
Piazza Comments & Criticisms

- Pros
  - Dependency acquisition modules are pluggable
  - Fault graphs serve as intuitive models
  - Useful for people who have no prior knowledge of correlated failures in system
- Cons
  - Only considers static dependencies
  - Failure probabilities, required by INDaaS, may be difficult to obtain, and their accuracy is questionable
  - Cloud providers may not have enough incentives to share data
  - Auditing is time-consuming

Thank you!

BACKUP SLIDES

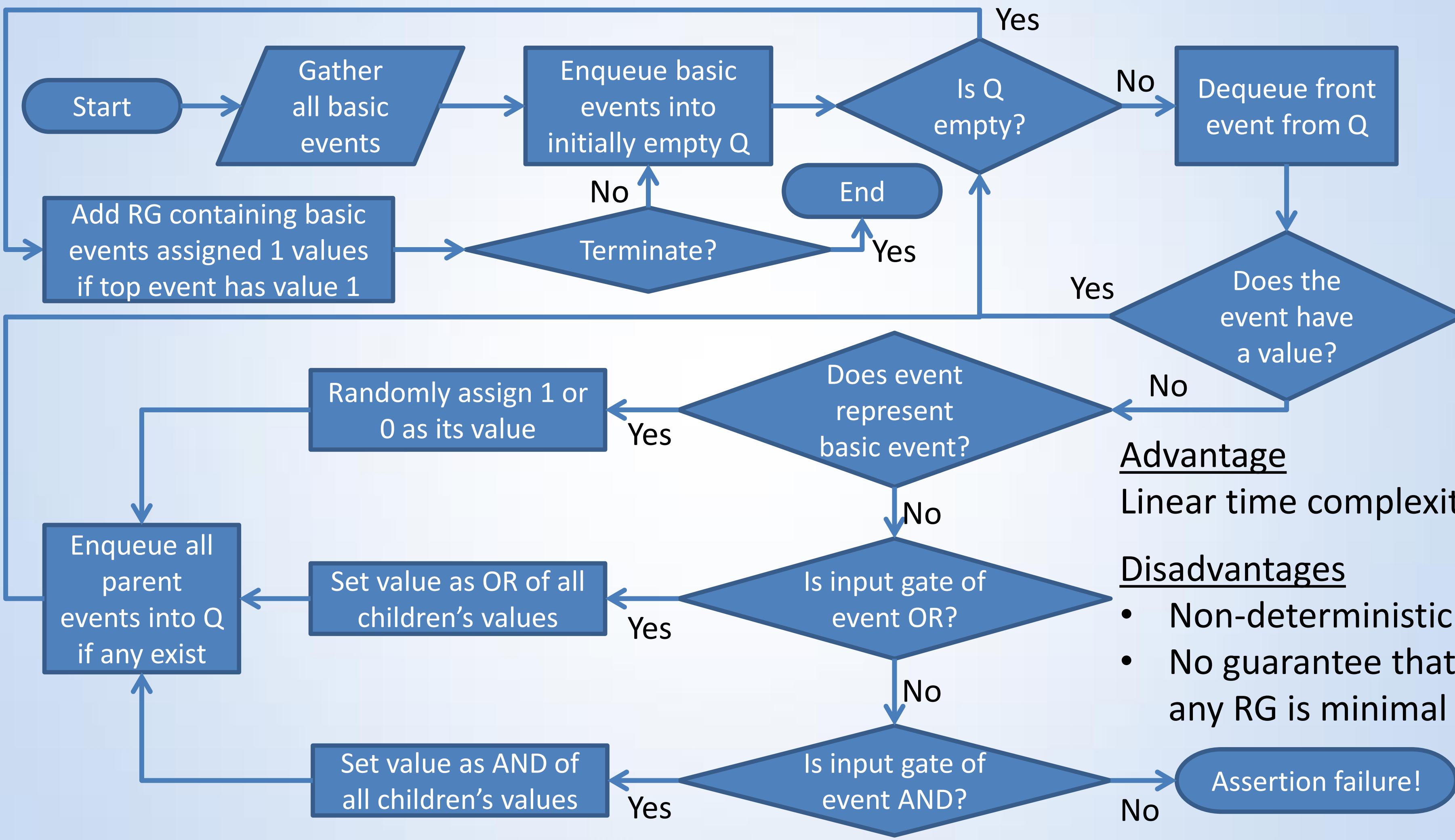
Minimal RG Algorithm



<u>Advantage</u>
Results are exact.

<u>Disadvantage</u>
Algorithm is NP hard!

Failure Sampling Algorithm



<u>Advantage</u>

Linear time complexity

<u>Disadvantages</u>

- Non-deterministic
- No guarantee that any RG is minimal