Privacy-Preserving Reasoning

Ruzica Piskac

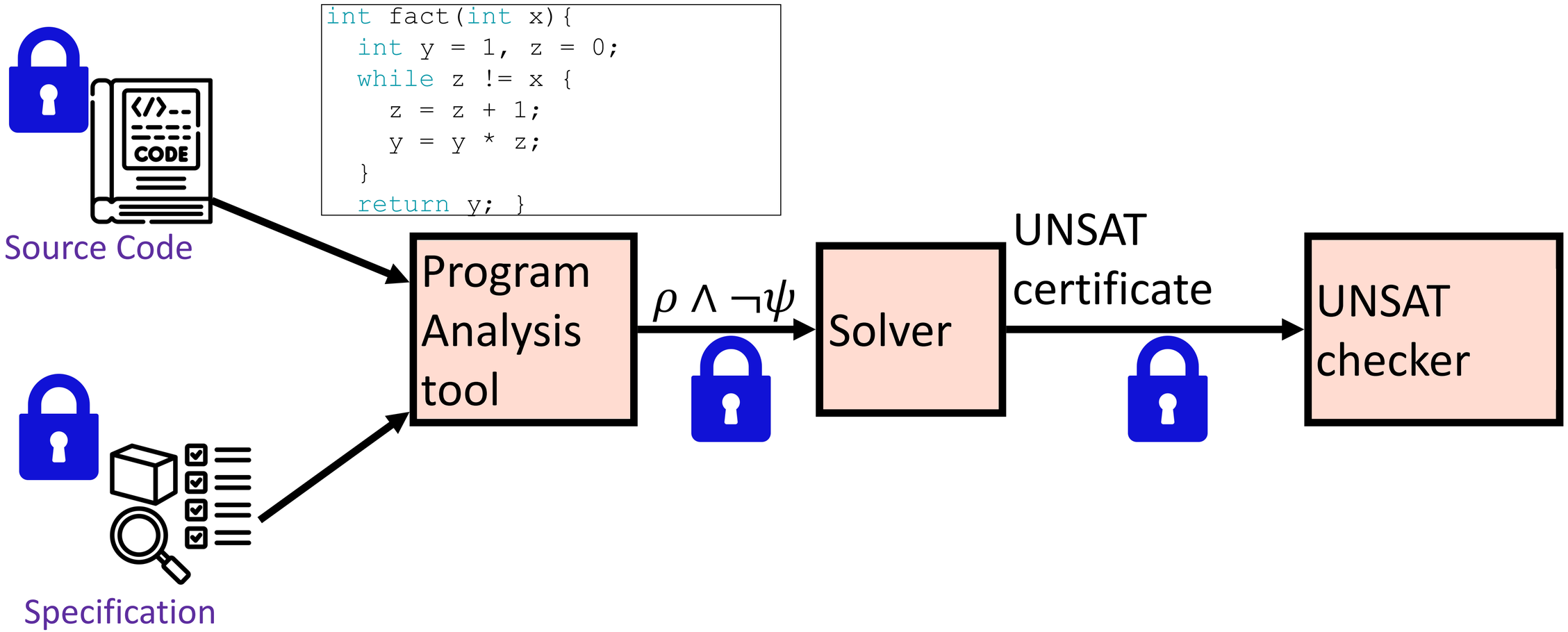
Yale University

Fifteenth Summer School on Formal Techniques
May 26, 2026

Overview

- Privacy-preserving ZK-SMT proof verification
  - ZK proofs
  - Resolution proofs
  - Verifying ZK-UNSAT in Boolean logic
  - Verifying ZK SMT proofs
- Privacy-preserving SAT solver
  - DPLL algorithm
  - Secure multi-party computation
  - ppSAT
- Ou: efficient programming framework for ZK protocols
  - Language design
  - Compiler for cloud-ready ZK proofs
- Conclusions

Privacy-Preserving Formal Verification



Evaluation

- SV-COMP

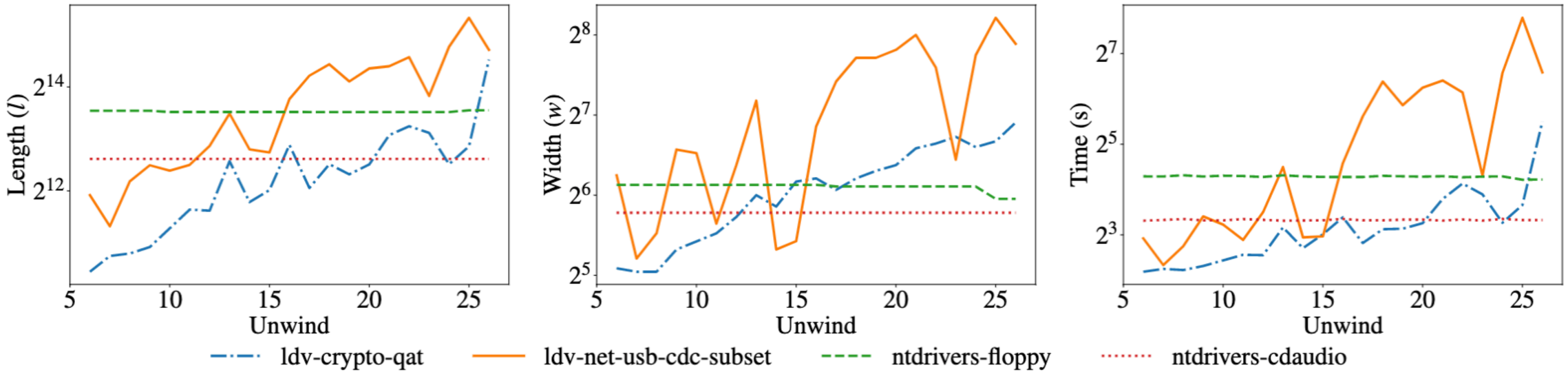
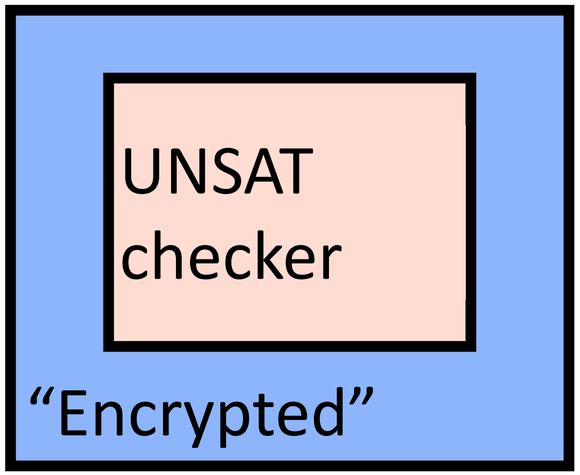


Figure 11: Verification features vs. bound on loop unwindings for drivers. Plots of refutation length, width, and verification time vs. bound on loop unwindings for a set of Windows NT and Linux drivers.

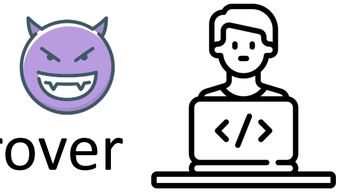
- Time and memory requirements depend on the length and clausal width of the proof
- ZKUNSAT takes less than min to verify proofs of 1 large width (400) and length (8000)

Privacy-Preserving Formal Verification

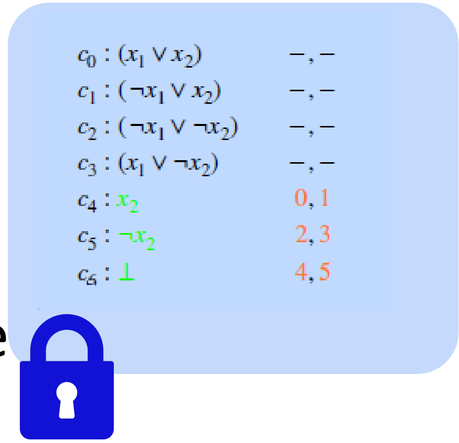


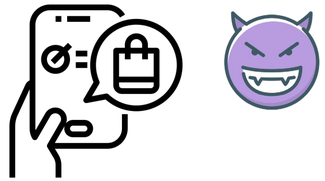
The First Step:

Verify UNSAT of Encrypted Formulae with Privacy Preserved



- Prove that ϕ is unsatisfiable
- Keep information on ϕ private

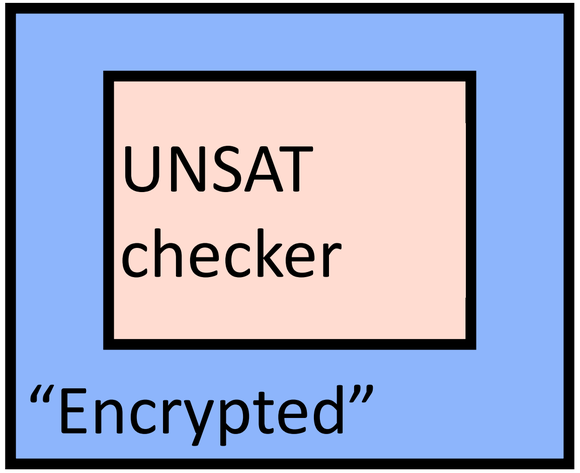




Verifier

- Validate prover’s claim about ϕ

Privacy-Preserving Formal Verification



Proving UNSAT in Zero Knowledge

Ning Luo
Yale University
ning.luo@yale.edu

Timos Antonopoulos
Yale University
timos.antonopoulos@yale.edu

William Harris
Galois, Inc.
wrharris@galois.com

Ruzica Piskac
Yale University
ruzica.piskac@yale.edu

Eran Tromer
Columbia University
et2555@columbia.edu

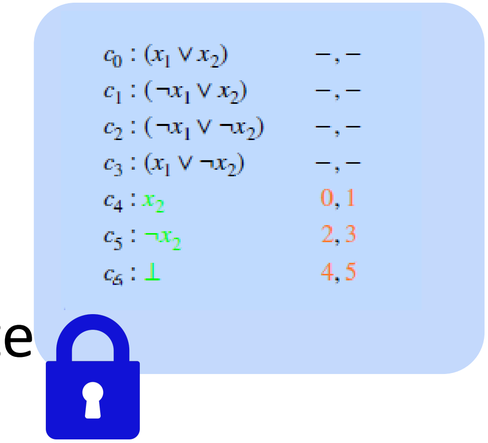
Xiao Wang
Northwestern University
wangxiao@cs.northwestern.edu

The First Step:

Verify UNSAT of Encrypted Formulae with Privacy Preserved

Prover

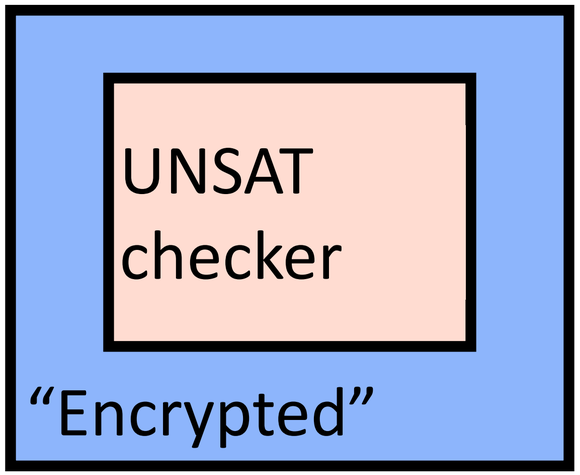
- Prove that ϕ is unsatisfiable
- Keep information on ϕ private



Verifier

- Validate prover’s claim about ϕ

Privacy-Preserving Formal Verification




```
1 extern int f(int x); // f(-x) = -f(x)
2
3 short f-cast(int a) {
4    int b = -a;
5    if (f(b) < SHRT_MIN || SHRT_MAX < f(b))
      error;
6    else return short (f(b));
7 }
```

The First Step:

Verify UNSAT of Encrypted Formulae with Privacy Preserved

Prover

- Prove that ϕ is unsatisfiable
- Keep information on ϕ private

SAT formula: 202 clauses

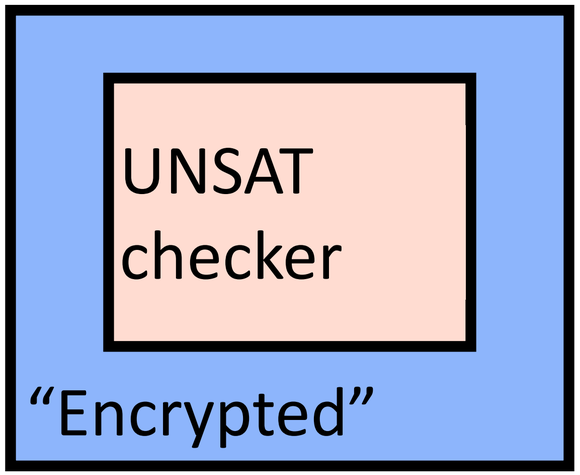
CNF SAT formula:

2024 clauses

Verifier

- Validate prover’s claim about ϕ

Privacy-Preserving Formal Verification



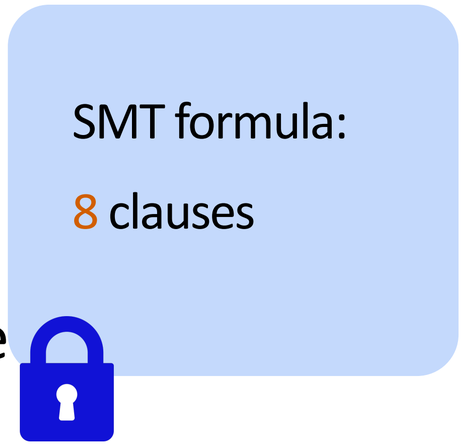

```
1 extern int f(int x); // f(-x) = -f(x)
2
3 short f-cast(int a) {
4    int b = -a;
5    if (f(b) < SHRT_MIN || SHRT_MAX < f(b))
      error;
6    else return short (f(b));
7 }
```

The First Step:

Verify UNSAT of Encrypted Formulae with Privacy Preserved

Prover

- Prove that ϕ is unsatisfiable
- Keep information on ϕ private



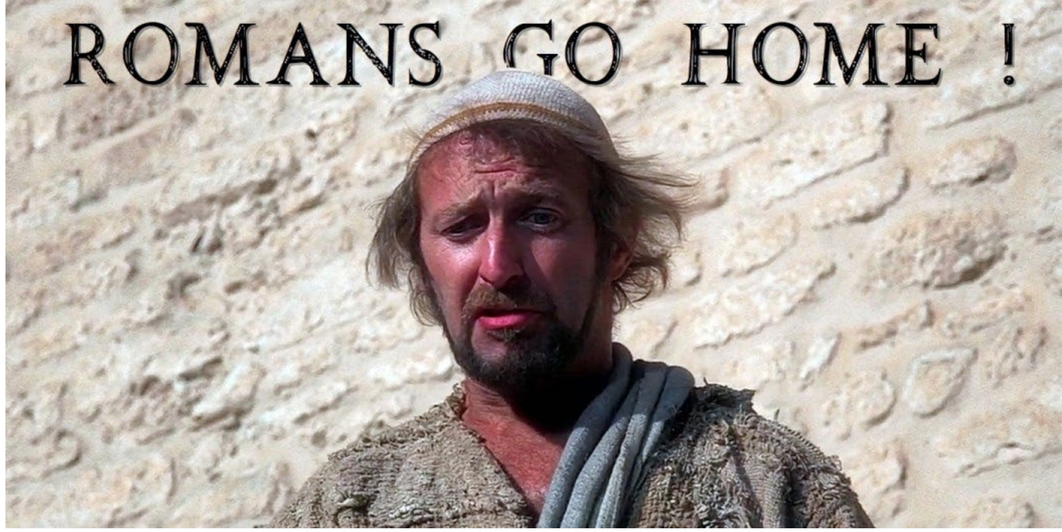
Verifier

- Validate prover’s claim about ϕ

Overview

- Privacy-preserving ZK-SMT proof verification
  - ZK proofs
  - Resolution proofs
  - Verifying ZK-UNSAT in Boolean logic
  - Verifying ZK SMT proofs
- Privacy-preserving SAT solver
  - DPLL algorithm
  - Secure multi-party computation
  - ppSAT
- Ou: efficient programming framework for ZK protocols
  - Language design
  - Compiler for cloud-ready ZK proofs
- Conclusions

SMT vs SAT: A Family Letter



---------- Forwarded message ---------
From: **Viktor Kunčak** <vkuncak@gmail.com>
Date: Wed, Sep 27, 2023 at 8:02 AM
Subject: My student wrote SAT instead of SMT
To: Ruzica Piskac <rpiskac@gmail.com>

so I told him he needs to write 100 times:

SAT solver is not good enough for program verification, we need SMT solvers

Now he used it as a git message for our course web page:

https://gitlab.epfl.ch/lara/cs550/-/commit/62717202ddaad5813d2692a344dce5baf1a4922d

Formats for Unsatisfiability of SAT/SMT

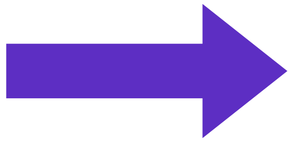
- DRAT: uses Resolution, for SAT
- SMT: combine Resolution with rules for theories
- Several formats:
  - LFSC
  - alethe
  - RESOLUTE

Verifying the Safety and Security of Private Programs

Satisfiability Modulo Theories (SMT) formulae:

- The Boolean formulae generalized to more complex theories
- Equality and uninterpreted function, linear integer arithmetic..

```
1 extern int f(int x); // f(-x) = -f(x)
2
3 short f-cast(int a) {
4    int b = -a;
5    if (f(b) < SHRT_MIN || SHRT_MAX < f(b))
      error;
6    else return short (f(b));
7 }
```



$$\begin{aligned}
&\texttt{SHRT_MAX} = 32767 \wedge \texttt{SHRT_MIN} = -32768 \\
&\wedge f(-a) = -f(a) \\
&\wedge b = -a \\
&\wedge (\neg(f(b) < \texttt{SHRT_MIN} \vee \texttt{SHRT_MAX} < f(b)) \vee err) \\
&\wedge\ (f(b) < \texttt{SHRT_MIN} \vee \texttt{SHRT_MAX} < f(b) \vee ret = f(b)) \\
&\wedge \neg err\ \wedge\ f(a) < \texttt{SHRT_MIN}
\end{aligned}$$

Verifying the Safety and Security of Private Programs

Satisfiability Modulo Theories (SMT) formulae:

- The Boolean formulae generalized to more complex theories
- Equality and uninterpreted function, linear integer arithmetic…
- A finite set of rules defining correct deduction
- **Unsatisfiability proof: a finite number of deductions**

$$\begin{aligned}&\texttt{SHRT\_MAX} = 32767 \wedge \texttt{SHRT\_MIN} = -32768\\ &\wedge f(-a) = -f(a)\\ &\wedge b = -a\\ &\wedge (\neg(f(b) < \texttt{SHRT\_MIN} \vee \texttt{SHRT\_MAX} < f(b)) \vee err)\\ &\wedge (f(b) < \texttt{SHRT\_MIN} \vee \texttt{SHRT\_MAX} < f(b) \vee ret = f(b))\\ &\wedge \neg err \wedge f(a) < \texttt{SHRT\_MIN}\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{}{f(b) = f(-a) \vee \neg(b = -a)}\,\text{(Congruence, EUF)} \qquad \dfrac{}{b = -a}\,\text{(Assumption)}}{f(b) = f(-a)}\,\text{(Resolution, Boolean)} \qquad \ldots}{\bot}\,\text{(Some proof rules)}$$

Example: Linear Integer Arithmetic

How to prove that $2x = 1$ is unsatisfiable in LIA?

- Farkas Lemma

Farkas' lemma — Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$. Then exactly one of the following two assertions is true:

1. There exists an $\mathbf{x} \in \mathbb{R}^n$ such that $\mathbf{Ax} = \mathbf{b}$ and $\mathbf{x} \geq 0$.
2. There exists a $\mathbf{y} \in \mathbb{R}^m$ such that $\mathbf{A}^\top \mathbf{y} \geq 0$ and $\mathbf{b}^\top \mathbf{y} < 0$.

- Simplex

The simplex algorithm operates on linear programs in the canonical form

maximize $\mathbf{c^T x}$

subject to $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq 0$

Example: Linear Integer Arithmetic

How to prove that $2x = 1$ is unsatisfiable in LIA?

- Farkas Lemma

Farkas' lemma — Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$. Then exactly one of the following two assertions is true:

1. There exists an $\mathbf{x} \in \mathbb{R}^n$ such that $\mathbf{Ax} = \mathbf{b}$ and $\mathbf{x} \geq 0$.
2. There exists a $\mathbf{y} \in \mathbb{R}^m$ such that $\mathbf{A}^\top \mathbf{y} \geq 0$ and $\mathbf{b}^\top \mathbf{y} < 0$.

- Simplex

The simplex algorithm operates on linear programs in the canonical form

maximize $\mathbf{c^T x}$

subject to $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq 0$

Example: Linear Integer Arithmetic

How to prove that $2x = 1$ is unsatisfiable in LIA?

- Farkas Lemma

> **Farkas' lemma** — Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$. Then exactly one of the following two assertions is true:
>
> 1. There exists an $\mathbf{x} \in \mathbb{R}^n$ such that $\mathbf{Ax} = \mathbf{b}$ and $\mathbf{x} \geq 0$.
> 2. There exists a $\mathbf{y} \in \mathbb{R}^m$ such that $\mathbf{A}^\top \mathbf{y} \geq 0$ and $\mathbf{b}^\top \mathbf{y} < 0$.

- Simplex
  - Returns $x = \frac{1}{2}$
  - Gomory cut: $x \in \mathbb{Z} \leftrightarrow x \geq 1 \vee x \leq 0$
  - Thus: $2x = 1 \leftrightarrow 2x = 1 \wedge (x \geq 1 \vee x \leq 0)$
  - We need to check the satisfiability of $2x = 1 \wedge x \geq 1$and $2x = 1 \wedge x \leq 0$

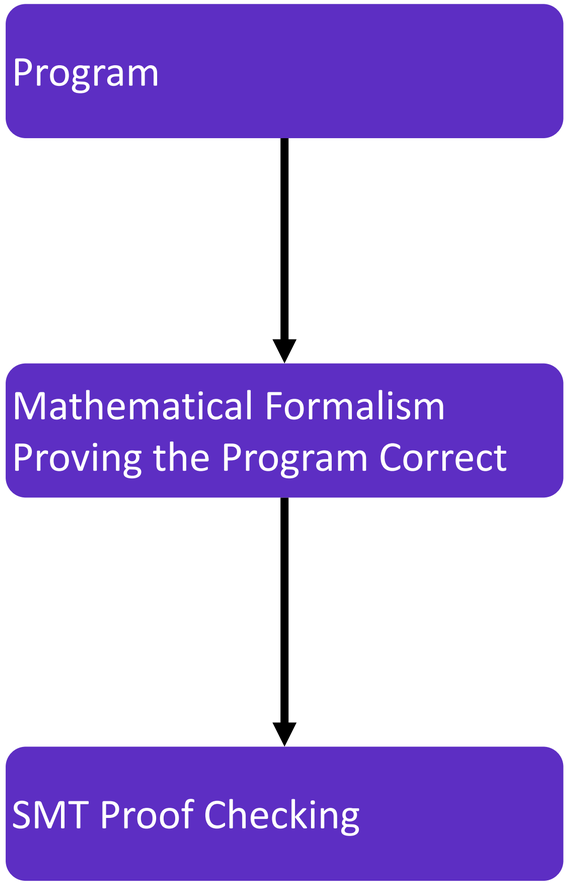
Formats for Unsatisfiability of SMT

How to prove that $2x = 1$ is unsatisfiable in LIA?

- Farkas Lemma
  - Formula $2x = 1 \wedge x \geq 1$ is unsatisfiable
  - Formula $2x = 1 \wedge x \leq 0$ is unsatisfiable
  - $(2x = 1 \wedge x \geq 1) \vee (2x = 1 \wedge x \leq 0)$ is unsatisfiable
  - $2x = 1$ is unsatisfiable

- Simplex
  - Returns $x = \frac{1}{2}$
  - Gomory cut: $x \in \mathbb{Z} \leftrightarrow x \geq 1 \vee x \leq 0$
  - Thus: $2x = 1 \leftrightarrow 2x = 1 \wedge (x \geq 1 \vee x \leq 0)$
  - We need to check the satisfiability of $2x = 1 \wedge x \geq 1$and $2x = 1 \wedge x \leq 0$

ZKSMT: An VM to Verify UNSAT as Program Execution

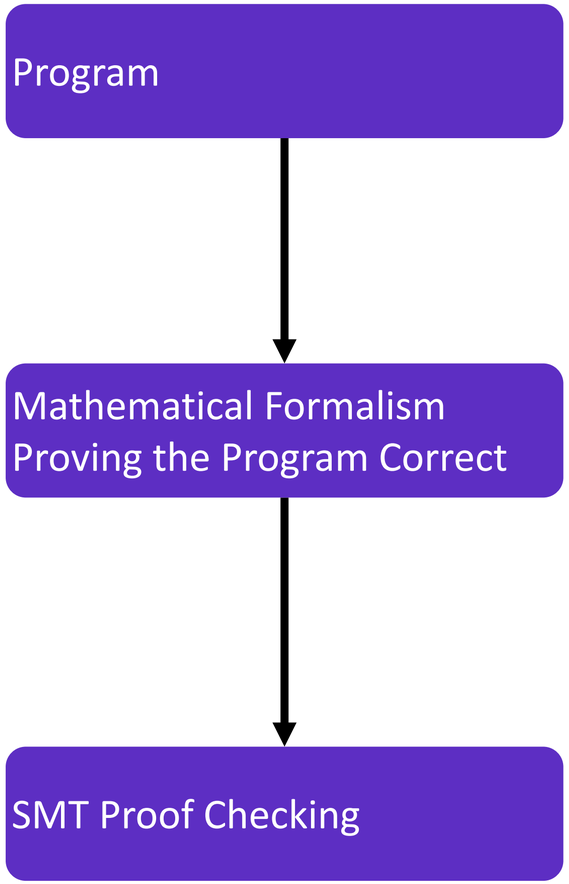


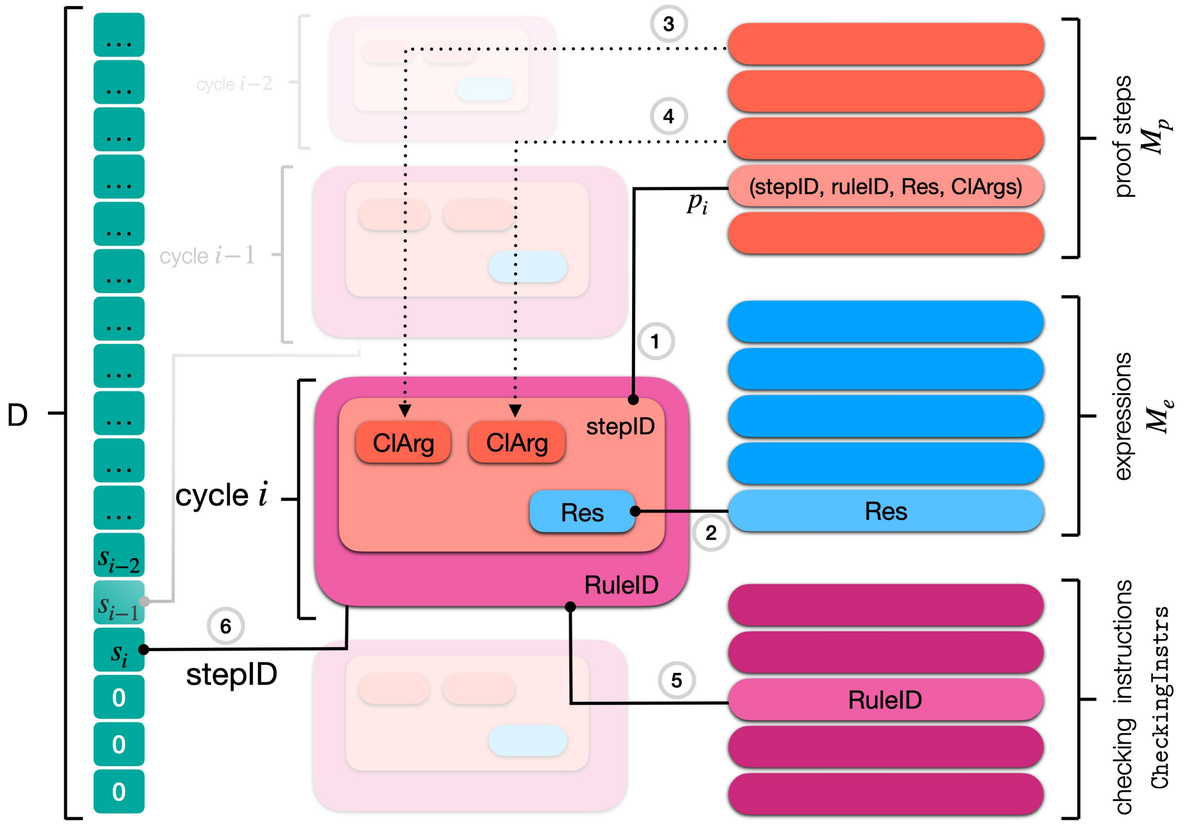
- It is difficult to build efficient ZKP checker for a generic FOL formula.
- Each SMT theory has its own verification techniques. A generic tool essentially prohibits theory-specific optimizations

Concurent work:

ZKPi: Proving Lean Theorems in Zero-Knowledge by Evan Laufer, Alex Ozdemir and Dan Boneh

ZKSMT: An VM to Verify UNSAT as Program Execution





ZKSMT Architecture

Tables

- Expressions store pointers to their children
- Steps store pointers to their premises and conclusion

Addr.	NodeID	ImmAddr	IndAddr	Meaning
&1	Var	ONE	{}	ONE
&2	Var	i	{}	i
&3	Var	a	{}	a
&4	Var	x	{}	x
&5	Apply	l	{&3}	$l(a)$
&6	Apply	g	{&3,&4}	$g(a,x)$
&7	Mul	0	{&1}	$0 * \texttt{ONE}$
&8	Mul	-1	{&1}	$-1 * \texttt{ONE}$
&9	Eq		{&2,&6}	$i = g(a,x)$
&10	Eq		{&7,&7}	$0 = 0$
&11	Leq		{&7,&8}	$0 \leq -1$
&12	Not		{&11}	$\neg(0 \leq -1)$
&13	Leq		{&7,&2}	$0 \leq i$
&14	Lt		{&2,&5}	$i < l(a)$
&15	And		{&13,&14}	$0 \leq i \wedge i < l(a)$
&16	Not		{&15}	$\neg(0 \leq i \wedge i < l(a))$

Table 1: Part of the expression table M_e for the proof of the safety of `find_and_replace`. We use & to denote the addresses of expressions.

StepID	RuleID	Premises	Result
#2	Assume		**&9**: $i = g(a,x)$
#3	Assume		**&13**: $0 \leq i$
#4	Assume		**&16**: $\neg(0 \leq i \wedge i < l(a))$
...			
#7	Res	{#5,#6}	$0-(-1)=1$
#8	Farkas	{#7}	**&12**: $\neg(0 \leq -1)$
...			
#10	Cong		$\neg((0 \leq i) = (0 \leq -1)) \vee \neg(0=0) \vee \neg(i=-1)$
#11	Res	{#9,#10}	$\neg(0=0) \vee \neg(i=-1)$
#12	Refl		**&10**: $0 = 0$
#13	Res	{#11,#12}	$\neg(i=-1)$
...			
#16	Res	{#14,#15}	**&15**: $0 \leq i \wedge i < l(a)$
#17	Res	{#4,#16}	False

Table 2: Part of the proof step array M_p for the proof of the safety of `find_and_replace`. Not all conclusions' addresses are shown. We use # to denote the IDs of proof steps.

Checking Instructions

- Most rules have no premises and no side conditions
- They introduce simple tautologies that more complex rules can use as premises later
- The checking instructions only need to perform simple pattern matching

RuleID	Conclusion
Boolean	
TruePos	$\bigvee\{\mathsf{True}\}$
FalseNeg	$\bigvee\{\neg\mathsf{False}\}$
ExclMid	$\bigvee\{\neg a, a\}$
ImplPos1	$\bigvee\{a \to b, a\}$
ImplPos2	$\bigvee\{a \to b, \neg b\}$
ImplNeg	$\bigvee\{\neg(a \to b), \neg a, b\}$
EquivPos1	$\bigvee\{a = b, a, b\}$
EquivPos2	$\bigvee\{a = b, \neg a, \neg b\}$
EquivNeg1	$\bigvee\{\neg(a = b), a, \neg b\}$
EquivNeg2	$\bigvee\{\neg(a = b), \neg a, b\}$
EUF	
Refl	$\bigvee\{a = a\}$
Symm	$\bigvee\{a = b, \neg(b = a)\}$
Trans	$\bigvee\{a = c, \neg(a = b), \neg(b = c)\}$
LIA	
Total	$\bigvee\{a \leq b, b < a\}$
Trichotomy	$\bigvee\{a < b, a = b, b < a\}$
AddSingle	$\sum\{a\} = a$
MulSingle	$1 * a = a$
MulDist	$c * (\sum_{i=0}^{n} d_i * x_i) = \sum_{i=0}^{n} cd_i * x_i$

Checking Instructions (Complex Rules)

- Premises and side conditions
- Angle brackets represent multisets
  - Order is irrelevant, but multiplicity matters
- Most of these rules take linear time to check
- We can provide extended witnesses to make checking faster

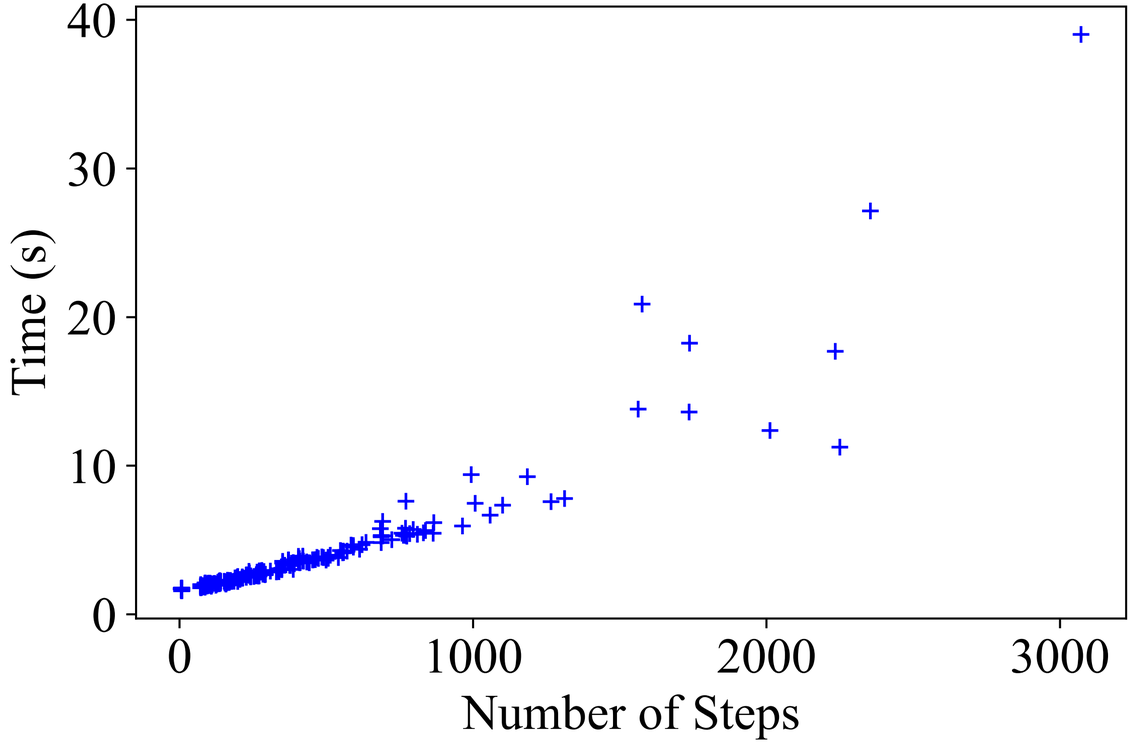
RuleID	Side Condition	Premises	Conclusion
	Boolean		
Resolution	$\exists p.p \in \langle A \rangle, \neg p \in \langle B \rangle,$ $\langle A \rangle \subseteq \langle C \rangle \uplus \langle p \rangle, \langle B \rangle \subseteq \langle C \rangle \uplus \langle \neg p \rangle$	$\bigvee A, \bigvee B$	$\bigvee C$
DeDup	$\forall a \in \langle A \rangle.\ a \in \langle B \rangle$	$\bigvee A$	$\bigvee B$
OrNil		$\bigvee \{\}$	False
OrSingle		a	$\bigvee \{a\}$
OrSingleRev		$\bigvee \{a\}$	a
AndPos	$\exists A, B. \langle \bigwedge A \rangle \uplus \langle B \rangle = \langle C \rangle, \bigwedge A = \bigwedge_{i=0}^{n} a_i, \bigvee B = \bigvee_{i=0}^{n} \neg a_i$		$\bigvee C$
AndNeg	$a \in \langle A \rangle$		$\bigvee \{\neg \bigwedge A, a\}$
OrPos	$a \in \langle A \rangle$		$\bigvee \{\bigvee A, \neg a\}$
OrNeg	$\exists A. \langle \neg \bigvee A \rangle \uplus \langle A \rangle = \langle B \rangle$		$\bigvee B$
	EUF		
Congruence	$\exists A, B, f. (fA = fB) \in C,\ \lvert A \rvert = \lvert B \rvert,$ $\forall i \in \{0, \ldots, \lvert A \rvert - 1\}. \neg (A_i = B_i) \in C$		$\bigvee C$
	LIA		
TotalInt	$i_0 = m * \texttt{ONE}, i_1 = (m+1) * \texttt{ONE}$		$\bigvee \{a \leq i_0, i_1 \leq a\}$
Consolidate	$\exists a, A_a, B_a, C.\ \langle A_a \rangle \uplus \langle C \rangle = \langle A \rangle, \langle B_a \rangle \uplus \langle C \rangle = \langle B \rangle,$ $A_a = \{\alpha_0 * a, \ldots, \alpha_{t-1} * a\}, B_a = \{\beta_0 * a, \ldots, \beta_{t'-1} * a\},$ $\alpha_0 + \cdots + \alpha_{t-1} = \beta_0 + \cdots + \beta_{t'-1}$		$\sum A = \sum B$
Flatten	$\exists \langle C \rangle, \langle D \rangle.\ \langle C \rangle \uplus \langle \sum D \rangle = \langle A \rangle,\ \langle C \rangle \uplus \langle D \rangle = \langle B \rangle$		$\sum A = \sum B$
Farkas	$\forall i \in \{0, \ldots, n\}.\ m_i \geq 0$ either $c > 0$, or $c = 0$ and $\exists j.\ \leq_j$ is $<$	$\sum_{i=0}^{n} (m_i * a_i) +$ $(-m_i * b_i) = c$	$\bigvee_{i=0}^{n} \{\neg (a_i \leq_i b_i)\}$

Zero-Knowledge Support

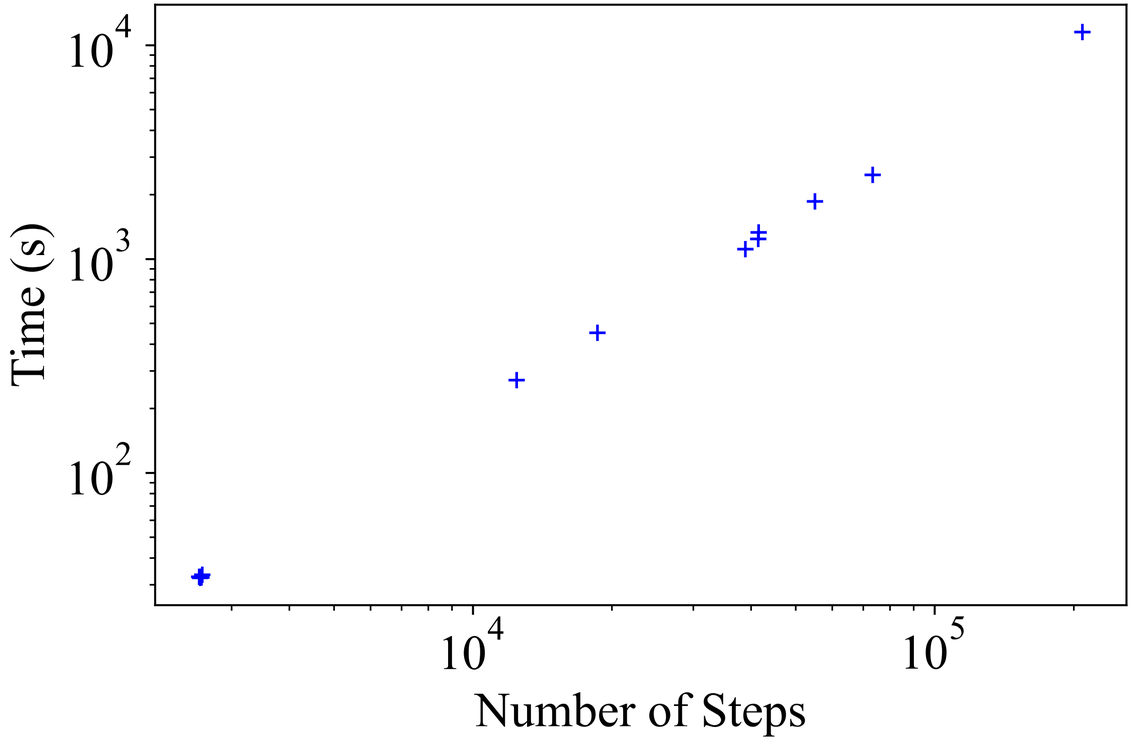
- We hide the values manipulated by each proof step
  - Different uses of the same rule are indistinguishable
- We use polynomial commitment over a finite field to hide most values in a proof
  - We can compare integers and perform arithmetic operations on them without revealing their values to the verifier
  - We can do the same thing for Boolean values

Evaluation: Boogie and WiSA Benchmarks

- The benchmarks can contain Boolean logic, EUF, LIA, or a combination of the three
- ZKSMT generally scales linearly as the number of proof steps increases
- Cheesecloth (generic ZK program evaluation) takes almost two hours for the smallest Boogie benchmark, but ZKSMT takes only two seconds



Boogie

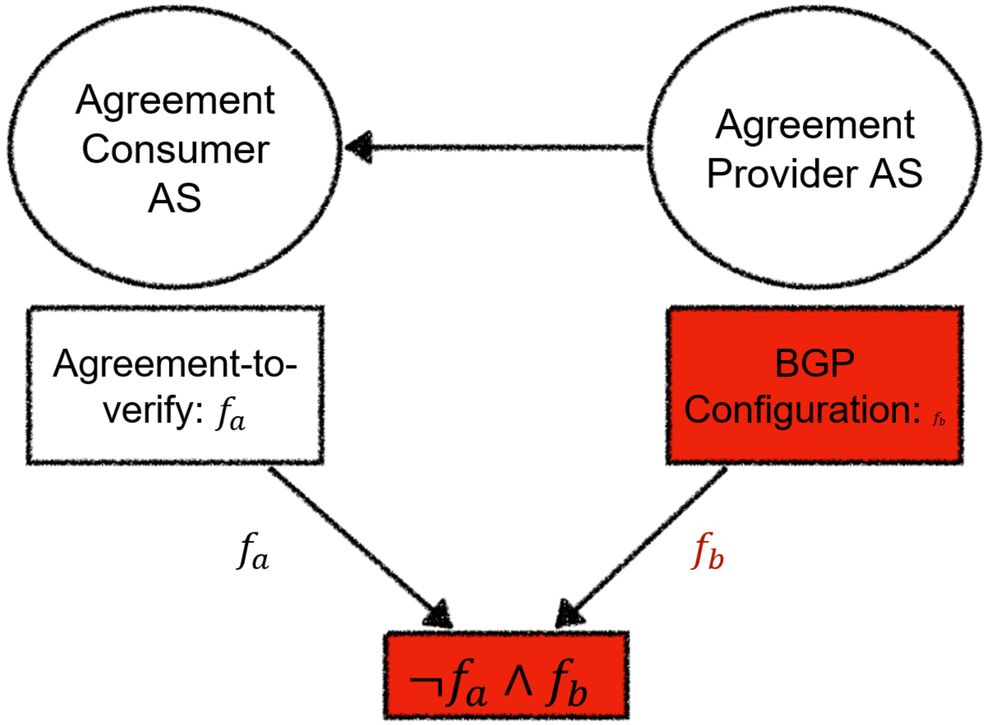


WiSA

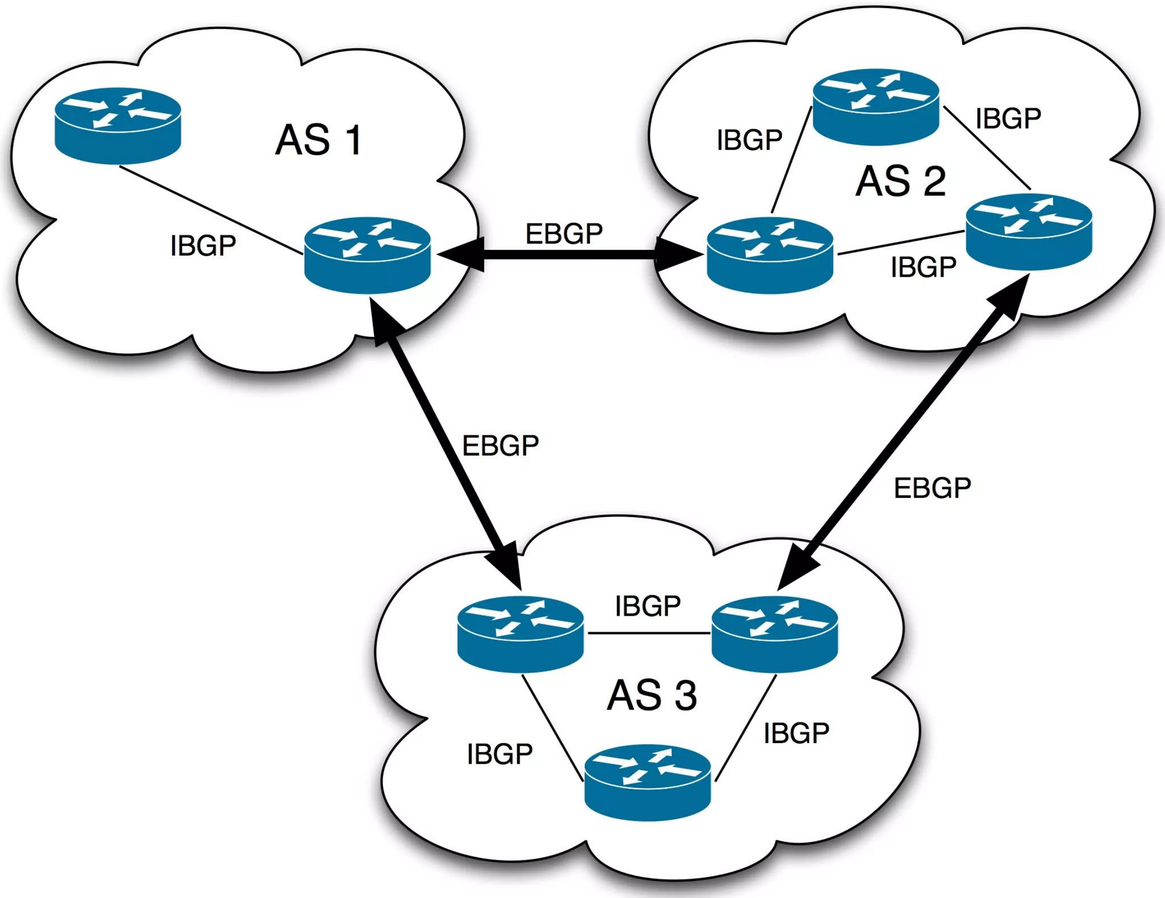
Overview

- Privacy-preserving ZK-SMT proof verification
  - ZK proofs
  - Resolution proofs
  - Verifying ZK-UNSAT in Boolean logic
  - Verifying ZK SMT proofs
- Privacy-preserving SAT solver
  - DPLL algorithm
  - Secure multi-party computation
  - ppSAT
- Ou: efficient programming framework for ZK protocols
  - Language design
  - Compiler for cloud-ready ZK proofs
- Conclusions

Why private programs ?



Private configuration in interdomain verification



Question: If $f_a \wedge f_b$ are private, can we still check their satisfiability?

Boolean SAT Solving

- **A Conjunction Normal Form (CNF) formula :**
  - Conjunction of disjunctions of literals
  - $(x_3 \vee x_0 \vee x_1) \wedge (\neg x_0 \vee x_2) \wedge (x_1 \vee x_2) \wedge \neg x_1 \wedge (\neg x_3 \vee \neg x_0) \wedge (\neg x_0 \vee x_3)$
- **Given a CNF formula ϕ**
  - SAT: there is a variable assignment $\mathcal{M}$ that makes ϕ evaluate to true
  - UNSAT: ϕ always evaluates to false
  - $\mathcal{M} = \{x_0 = False, x_1 = False, x_2 = True, x_3 = True\}$

Boolean SAT Solving: A DPLL Example

- Branch and backtrack algorithm
- Worst-case exponential
- [Davis at al, 1962]

The original DPLL procedure

- Tries to build incrementally a satisfying truth assignment M for a CNF formula F
- M is grown by
  - deducing the truth value of a literal from M and F, or
  - guessing a truth value
- If a wrong guess for a literal leads to an inconsistency, the procedure backtracks and tries the opposite value

The Original DPLL Procedure – Example

Comment:

To simplify notation, we denote formula

$(x_1 \vee x_2) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge x_1$

with

$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

The Original DPLL Procedure – Example

assign	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg\, 1 \vee \neg\, 2,\ \neg\, 1 \vee \neg\, 3 \vee \neg\, 4,\ 1$

Deduce 1

1	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg\, 1 \vee \neg\, 2,\ \neg\, 1 \vee \neg\, 3 \vee \neg\, 4,\ 1$

Deduce $\neg 2$

1, 2	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg\, 1 \vee \neg\, 2,\ \neg\, 1 \vee \neg\, 3 \vee \neg\, 4,\ 1$

Guess 3

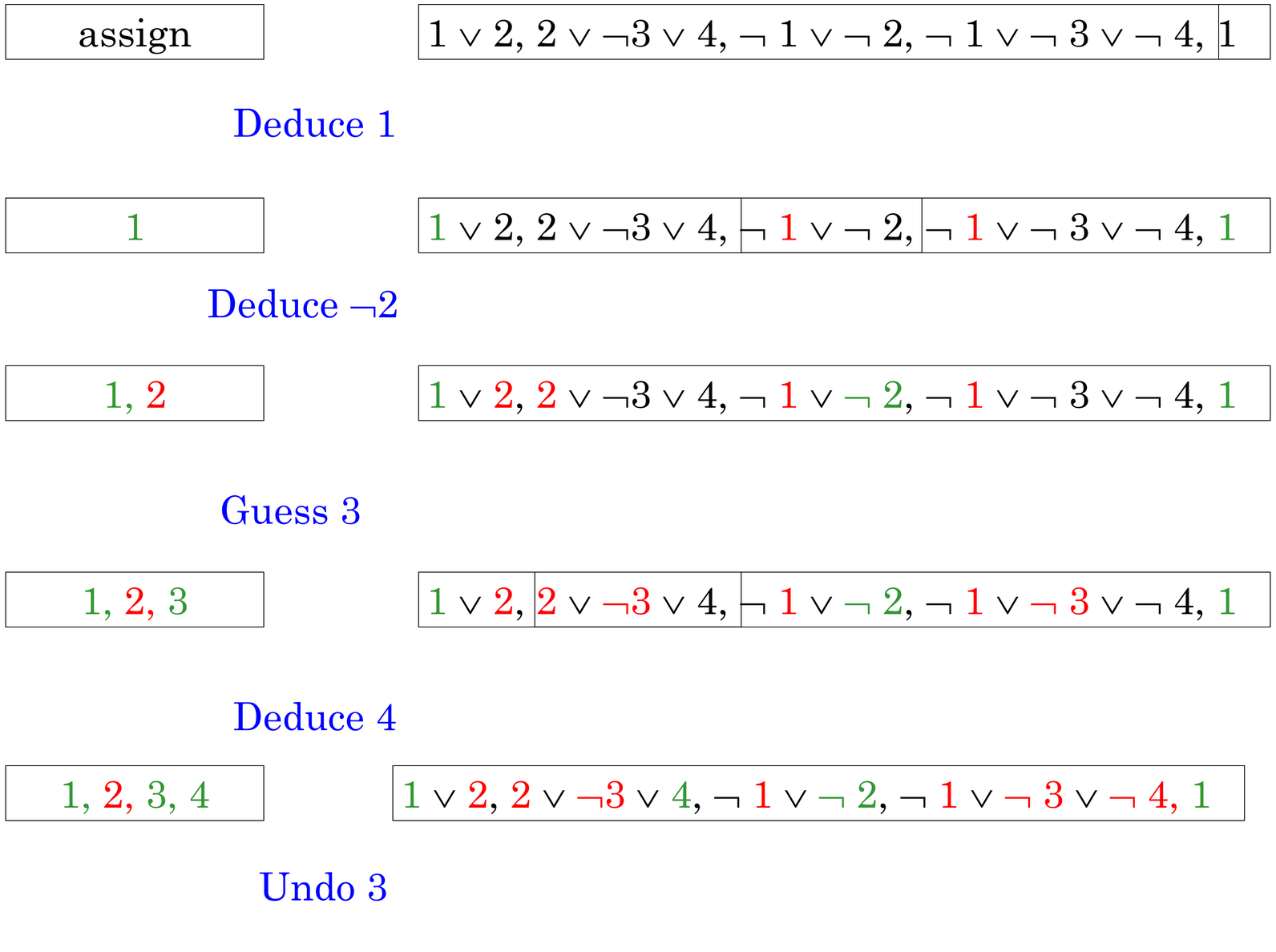
1, 2, 3	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg\, 1 \vee \neg\, 2,\ \neg\, 1 \vee \neg\, 3 \vee \neg\, 4,\ 1$

Deduce 4

1, 2, 3, 4	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg\, 1 \vee \neg\, 2,\ \neg\, 1 \vee \neg\, 3 \vee \neg\, 4,\ 1$

Conflict

The Original DPLL Procedure – Example



The Original DPLL Procedure – Example

assign	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg\ 1 \vee \neg\ 2,\ \neg\ 1 \vee \neg\ 3 \vee \neg\ 4,\ 1$

Deduce 1

1	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg\ 1 \vee \neg\ 2,\ \neg\ 1 \vee \neg\ 3 \vee \neg\ 4,\ 1$

Deduce $\neg 2$

1, 2	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg\ 1 \vee \neg\ 2,\ \neg\ 1 \vee \neg\ 3 \vee \neg\ 4,\ 1$

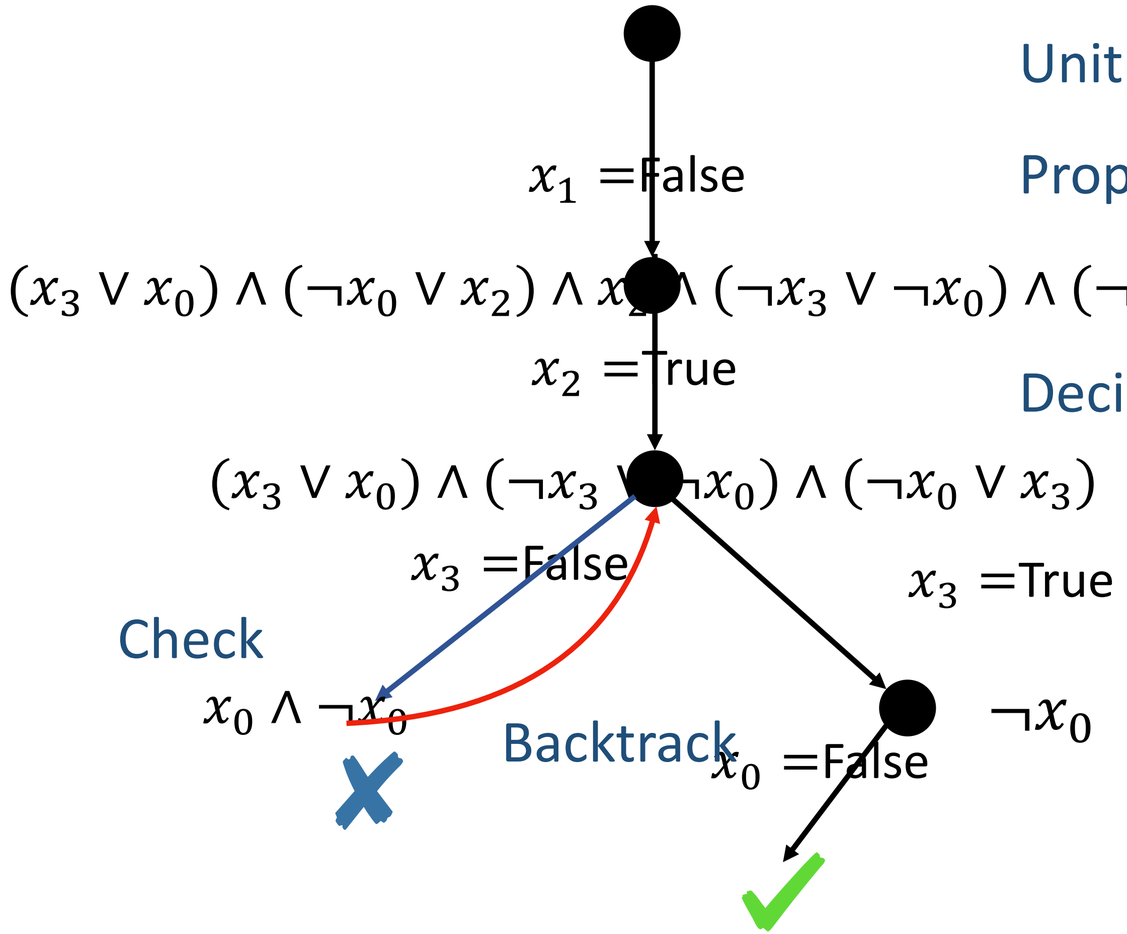
Deduce $\neg 3$

1, 2, 3	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg\ 1 \vee \neg\ 2,\ \neg\ 1 \vee \neg\ 3 \vee \neg\ 4,\ 1$

Model Found

Boolean SAT Solving: A DPLL Example

$$\phi(x_0, x_1, x_2, x_3) = (x_3 \vee x_0 \vee x_1) \wedge (\neg x_0 \vee x_2) \wedge (x_1 \vee x_2) \wedge \neg x_1 \wedge (\neg x_3 \vee \neg x_0) \wedge (\neg x_0 \vee x_3)$$



Cryptographic preliminaries

- Garbled circuit [Yao, 86]
  - Enables 2-party secure computation
  - Evaluate $F(x, y)$ over their private inputs x and y
- Oblivious stack [Zahur & Evans, 2013] [Wang et al, 2014]
  - Conditional operations
  - A secret Boolean variable that indicates whether the operation actually occurs
  - Private operations: pop / push

Example of a 2-party secure computation

x: 0010101010

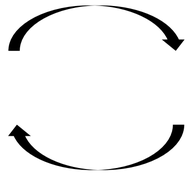
y: 1100010100

- Alice has a secret value x, Bob has a secret value y
- Jointly they need to compute $x \oplus y$

Example of a 2-party secure computation

x: 0010101010
Ra:1100110010

Ra



Rb

y: 1100010100
Rb:010011100

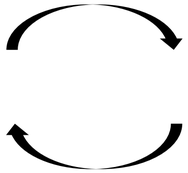
- Alice has a secret value x, Bob has a secret value y
- Jointly they need to compute x ⊕ y
- Alice sends to Bob random value Ra
- Bob sends to Alice random value Rb

Example of a 2-party secure computation

x: 0010101010
Ra:1100110010

Ra

y: 1100010100
Rb:010011100



x ⊕ Ra ⊕ Rb:
101010100

Rb

y ⊕ Ra ⊕ Rb:
010011010

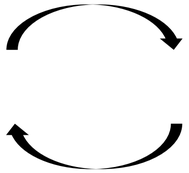
- Alice has a secret value x, Bob has a secret value y
- Jointly they need to compute x ⊕ y
- Alice sends to Bob random value Ra
- Bob sends to Alice random value Rb
- Alice computes x ⊕ Ra ⊕ Rb
- Bob computes y ⊕ Ra ⊕ Rb

Example of a 2-party secure computation

x: 0010101010
Ra:1100110010

Ra

y: 1100010100
Rb:010011100



Rb

x ⊕ Ra ⊕ Rb:
101010100

y ⊕ Ra ⊕ Rb:
010011010

x ⊕ Ra ⊕ Rb ⊕ y ⊕ Ra ⊕ Rb = x ⊕ y

- Alice has a secret value x, Bob has a secret value y
- Jointly they need to compute x ⊕ y
- Alice sends to Bob random value Ra
- Bob sends to Alice random value Rb
- Alice computes x ⊕ Ra ⊕ Rb
- Bob computes y ⊕ Ra ⊕ Rb
- They send those number to the third party who can easily compute x ⊕ y

Universality of Secure Multi-Party Computation

- **Theorem**
- Every *polynomial−time computable function can be securely computed via MPC.*

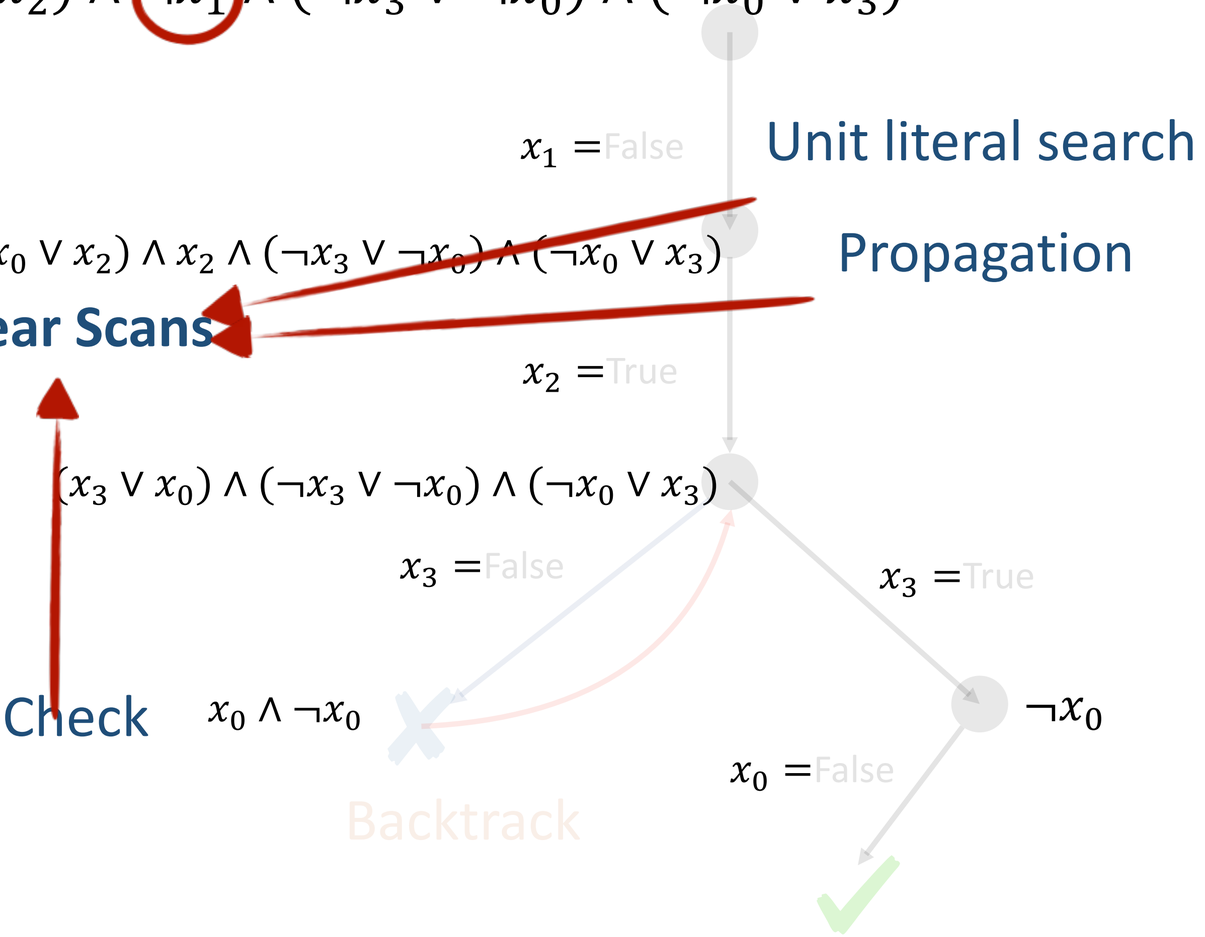
Classic References

- A. Yao, *Protocols for Secure Computations*, FOCS 1982
- O. Goldreich, S. Micali, A. Wigderson,
  How to Play ANY Mental Game, STOC 1987

Boolean SAT Solving: A DPLL Example

$$\phi(x_0, x_1, x_2, x_3)$$
$$= (x_3 \vee x_0 \vee x_1) \wedge (\neg x_0 \vee x_2) \wedge (x_1 \vee x_2) \wedge \neg x_1 \wedge (\neg x_3 \vee \neg x_0) \wedge (\neg x_0 \vee x_3)$$

$$\begin{array}{c} \\ x_0 \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{cccccc} c_0, & c_1, & c_2, & c_3, & c_4, & c_5 \end{array} \\ \begin{bmatrix} 1 & -1 & 0 & 0 & -1 & -1 \\ 1 & 0 & 1 & -1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \end{array}$$



Linear Scan in MPC

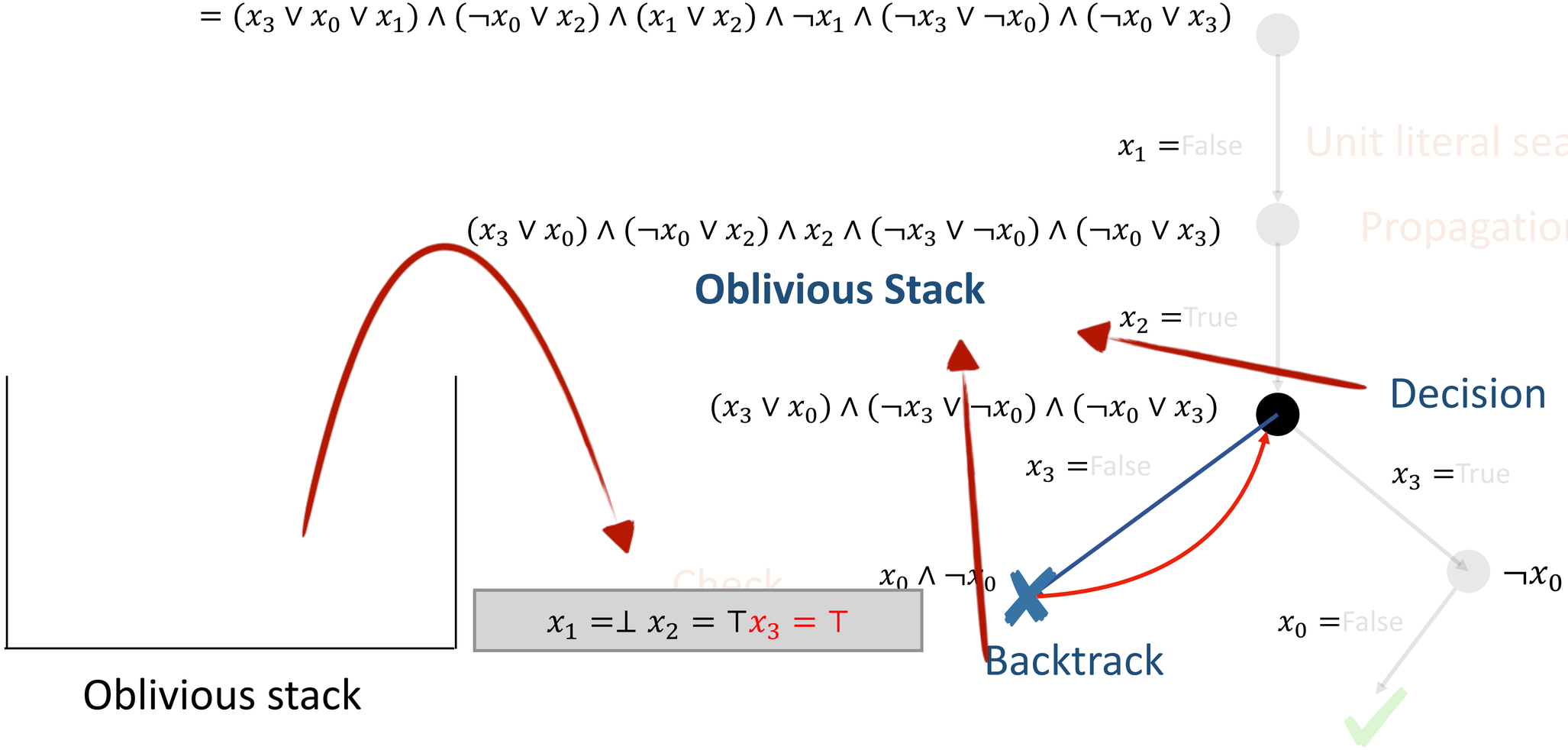
- Sequentially processes secret-shared data item by item
- Commonly used for searching, filtering, and aggregation over private inputs
- Complexity is typically $O(n)$,where $n$is the number of elements
- Frequently used in privacy-preserving databases, PSI, ORAM-free protocols, and secure analytics

- **Example:**
  Find whether a secret value $x$appears in a private list $(a_1, \ldots, a_n)$by securely computing:

$$\bigvee_{i=1}^{n} (x = a_i)$$

Boolean SAT Solving: A DPLL Example

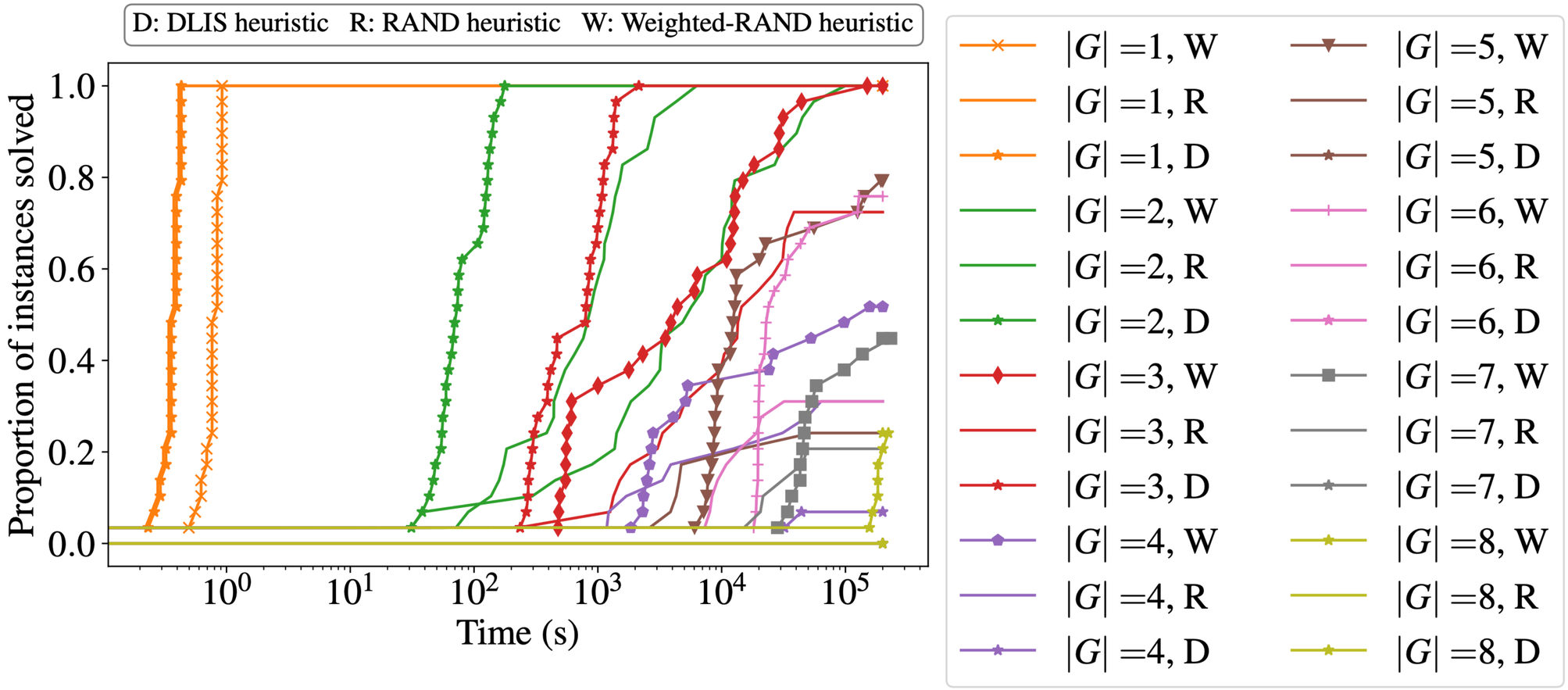
$$\phi(x_0, x_1, x_2, x_3)$$
$$= (x_3 \vee x_0 \vee x_1) \wedge (\neg x_0 \vee x_2) \wedge (x_1 \vee x_2) \wedge \neg x_1 \wedge (\neg x_3 \vee \neg x_0) \wedge (\neg x_0 \vee x_3)$$



Evaluation Results: Testbed

- **EMP-toolkit**
- **8GB of RAM**
- **Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz * 6 processor,**
- **Network bandwidth up to 10 Gbps**

Evaluation Results : HapMap Dataset



- 232 SAT instances in total
- $|G|$ is the parameter of instances in HapMap
- When $|G| = 8$, formulae have 900 variables and 1000 clauses

Overview

- Privacy-preserving ZK-SMT proof verification
  - ZK proofs
  - Resolution proofs
  - Verifying ZK-UNSAT in Boolean logic
  - Verifying ZK SMT proofs
- Privacy-preserving SAT solver
  - DPLL algorithm
  - Secure multi-party computation
  - ppSAT
- Ou: efficient programming framework for ZK protocols
  - Language design
  - Compiler for cloud-ready ZK proofs
- Conclusions

Zero-Knowledge Proofs

[Goldwasser, Micali, Rackoff, 1985]

Goldwasser & Micali - Turing award 2012

Correct and **private** computations are needed **everywhere**

(

Authentication Systems

Smart contracts

(e.g. ZK-rollups for computation efficiency)

zkApps

Privacy in digital currencies

(e.g. Zcash)

)

Zero-Knowledge Proofs

[Goldwasser, Micali, Rackoff, 1985]

Goldwasser & Micali - Turing award 2012

Correct and **private** computations are needed **everywhere**

(

Authentication Systems

Smart contracts
(e.g. ZK-rollups for computation efficiency)

zkApps

Privacy in digital currencies
(e.g. Zcash)

)

… but they are **slow**!!!!

i.e. **2500 times slower** than plaintext

Overcoming Obstacles

Hype Cycle for Privacy, 2022 by **Gartner**

ZKPs remain in an **emerging state**. They still require a **common framework for applications** to leverage.

Only a limited number of practical implementations have emerged till date.

ZKPs require integration into applications. Downstream applications, such as CRMs and databases, will need some modification.

The variety of methodologies and the multiplicity of approaches to data management inhibit adoption. ZKPs will need to scale at the rate of blockchain transactional volumes in order to be effective.

Overcoming Obstacles

Hype Cycle for Privacy, 2022 by **Gartner**

ZKPs remain in an **emerging state**. They still require a **common framework for applications** to leverage.

Only a limited number of practical implementations have been created.

Too hard to use

ZKPs require integration into applications. Downstream applications, such as CRMs and databases, will need some modification.

The variety of methodologies and the multiplicity of approaches to data management inhibit adoption. ZKPs will need to scale at the rate of blockchain transactional volumes in order to be effective.

Too slow to use

Overcoming Obstacles

Hype Cycle for Privacy, 2022 by **Gartner**

ZKPs remain in an **emerging state**. They still require a **common framework for applications** to leverage.

Only a limited number of practical implementations have been created.

Too hard to use

ZKPs require integration into applications. Downstream applications, such as CRMs and databases, will need some modification.

```
pvt int[m][n] A = init_A();
pvt int[n][p] B = init_B();
pvt int[m][p] C = init_C();

for int t = 0; t < 10; t++ {
// test 10 times
 int[p] v;
 for int i = 0; i < p; i++ {
   v[i] = verifier_rand(0, 99);
 }
 pvt int[n] x = pvt_mul_pub(B, v);
 pvt int[m] y = pvt_mul_pvt(A, x);
 pvt int[m] z = pvt_mul_pub(C, v);

 for int i = 0; i < m; i++ {
   assert y[i] == z[i];
 }
}
```

The variety of methodologies and the multiplicity of approaches to data management inhibit adoption. ZKPs will need to scale at the rate of blockchain transactional volumes in order to be effective.

Too slow to use

Overcoming Obstacles

Hype Cycle for Privacy, 2022 by **Gartner**

Too hard to use

ZKPs remain in an **emerging state**. They still require a **common framework for applications** to leverage.

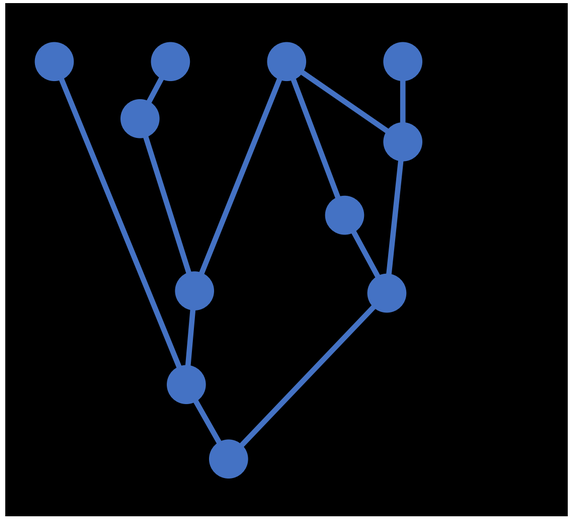
Only a limited number of practical implementations have been created.

ZKPs require integration into applications. Downstream applications, such as CRMs and databases, will need some modification.

Too slow to use

The variety of methodologies and the multiplicity of approaches to data management inhibit adoption. ZKPs will need to scale at the rate of blockchain transactional volumes in order to be effective.

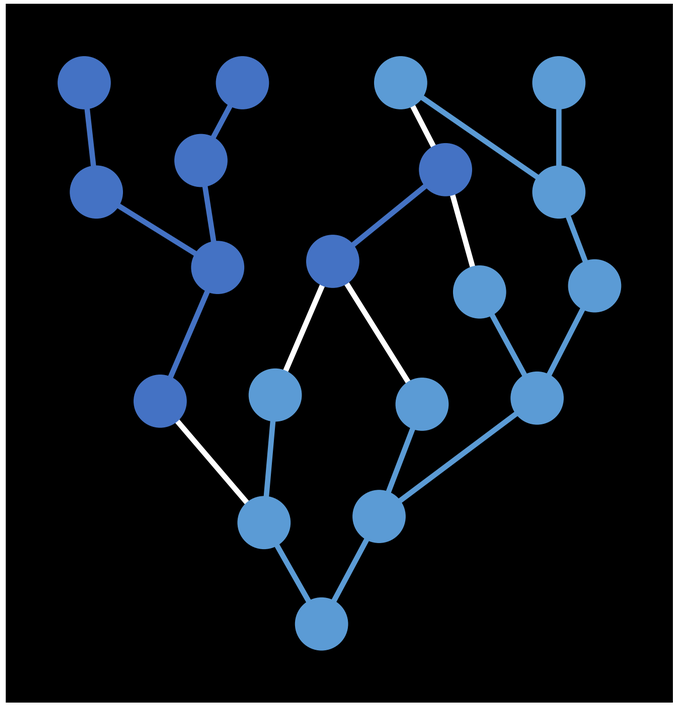
```
pvt int[m][n] A = init_A();
pvt int[n][p] B = init_B();
pvt int[m][p] C = init_C();

for int t = 0; t < 10; t++ {
// test 10 times
 int[p] v;
 for int i = 0; i < p; i++ {
  v[i] = verifier_rand(0, 99);
 }
 pvt int[n] x = pvt_mul_pub(B, v);
 pvt int[m] y = pvt_mul_pvt(A, x);
 pvt int[m] z = pvt_mul_pub(C, v);

 for int i = 0; i < m; i++ {
  assert y[i] == z[i];
 }
}
```



Our Solution

Programming Language

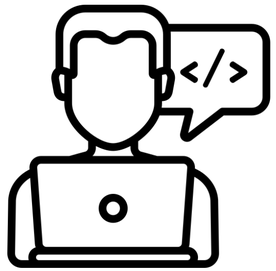
A high-level programming language that is familiar enough to novices, but also provides advanced features to be exploited by experts.



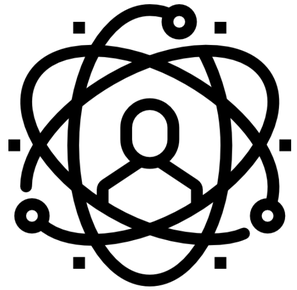
Compiler and Partitioner

A toolchain that first compiles the high-level program down to a circuit, and then finds the **optimal way to partition** that circuit into chunks that are then deployed to available commodity CPUs.

C-like General-Purpose ZK Frontend Language



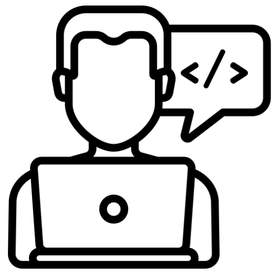
Normal Developers



Cryptography Experts

C-like General-Purpose ZK Frontend Language

Develop and compile the program as if using a traditional C compiler



Normal developers

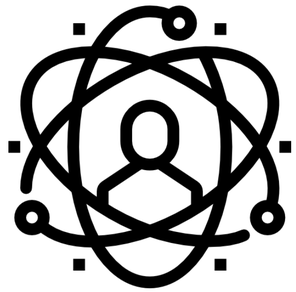
- Similar to C, has typical C features: array, struct, pointer, loop and etc
  - Not provided by most popular ZK languages
- No requirement on ZKP or cryptography
  - Intuitive type system for ZKP
  - Only need to know what variables are private

```
#define N 8
pvt1 int sum(pvt1 int[N] *a) {
  pvt1 int s = 0;
  for int i = 0; i < N; i = i+1; {
    s = s + a->[i];
  }
  return s;
}

void main() {
  pvt1 int[N] a = init();
  pvt1 int sum_a = sum(&a);
  assert(sum_a == 40);
  return;
}
```

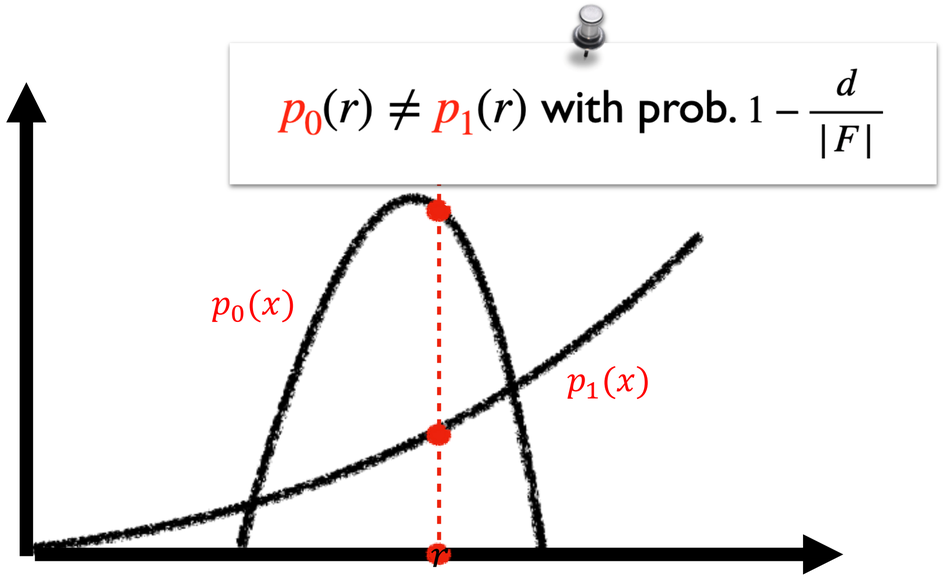
C-like General-Purpose ZK Frontend Language

Crypto experts can improve Ou program using their insights.

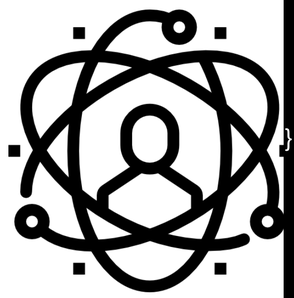
- Random challenge in run time
  - Commonly used in optimizing some interactive ZK protocols.



Cryptography Experts



C-like General-Purpose ZK Frontend Language



Cryptography

```
pvt int[m][n] A = init_A();
pvt int[n][p] B = init_B();
pvt int[m][p] C = init_C();

for int i = 0; i < m; i++ {
 for int j = 0; j < p; j++ {
  pvt int e = 0;
  for int k = 0; k < n; k++ {
   e = e + A[i][k] * B[k][j];
  }
  assert e == C[i][j];
 }
}
```

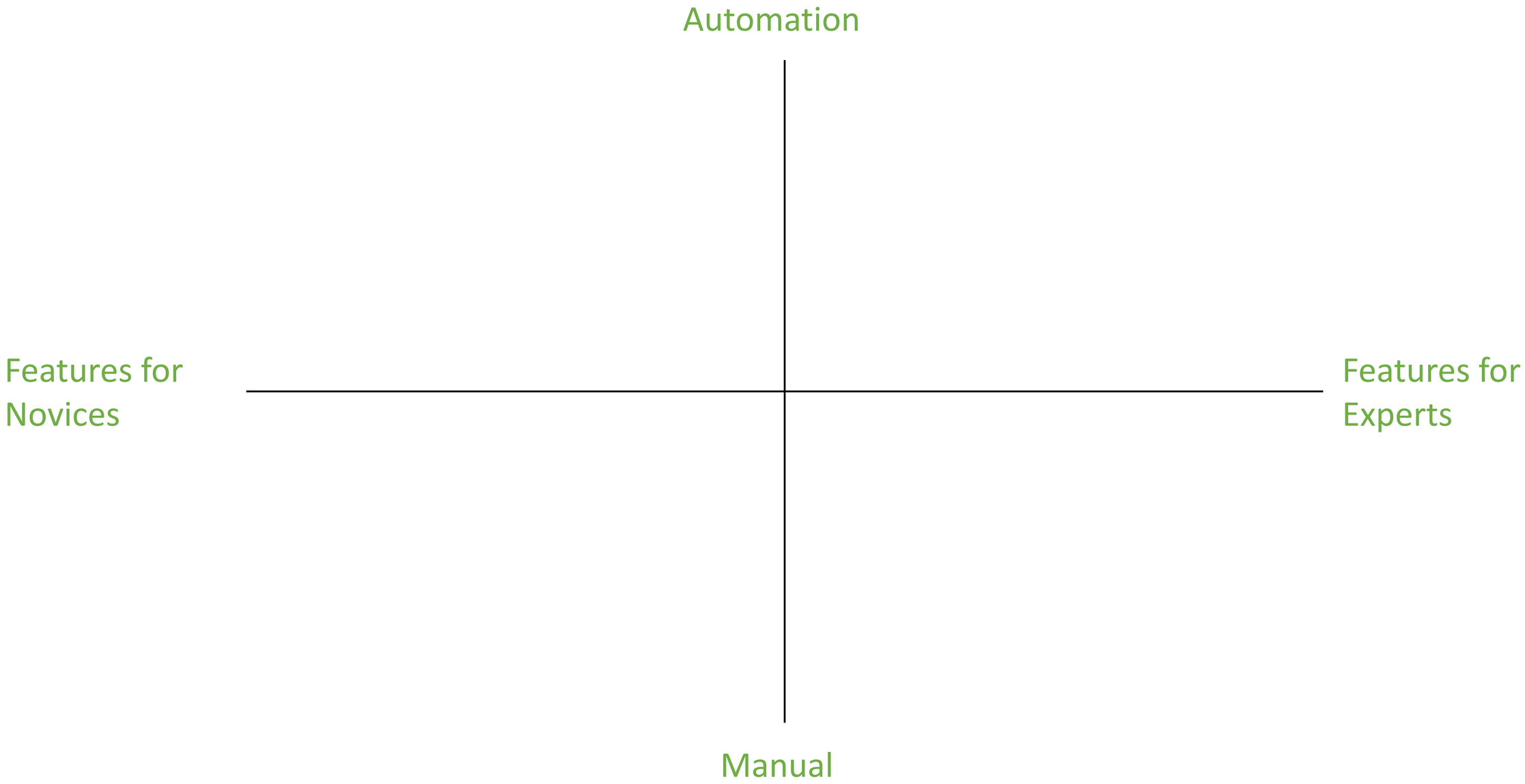
n^3

```
pvt int[m][n] A = init_A();
pvt int[n][p] B = init_B();
pvt int[m][p] C = init_C();

for int t = 0; t < 10; t++ {
// test 10 times
 int[p] v;
 for int i = 0; i < p; i++ {
  v[i] = verifier_rand(0, 99);
 }
 pvt int[n] x = pvt_mul_pub(B, v);
 pvt int[m] y = pvt_mul_pvt(A, x);
 pvt int[m] z = pvt_mul_pub(C, v);

 for int i = 0; i < m; i++ {
  assert y[i] == z[i];
 }
}
```

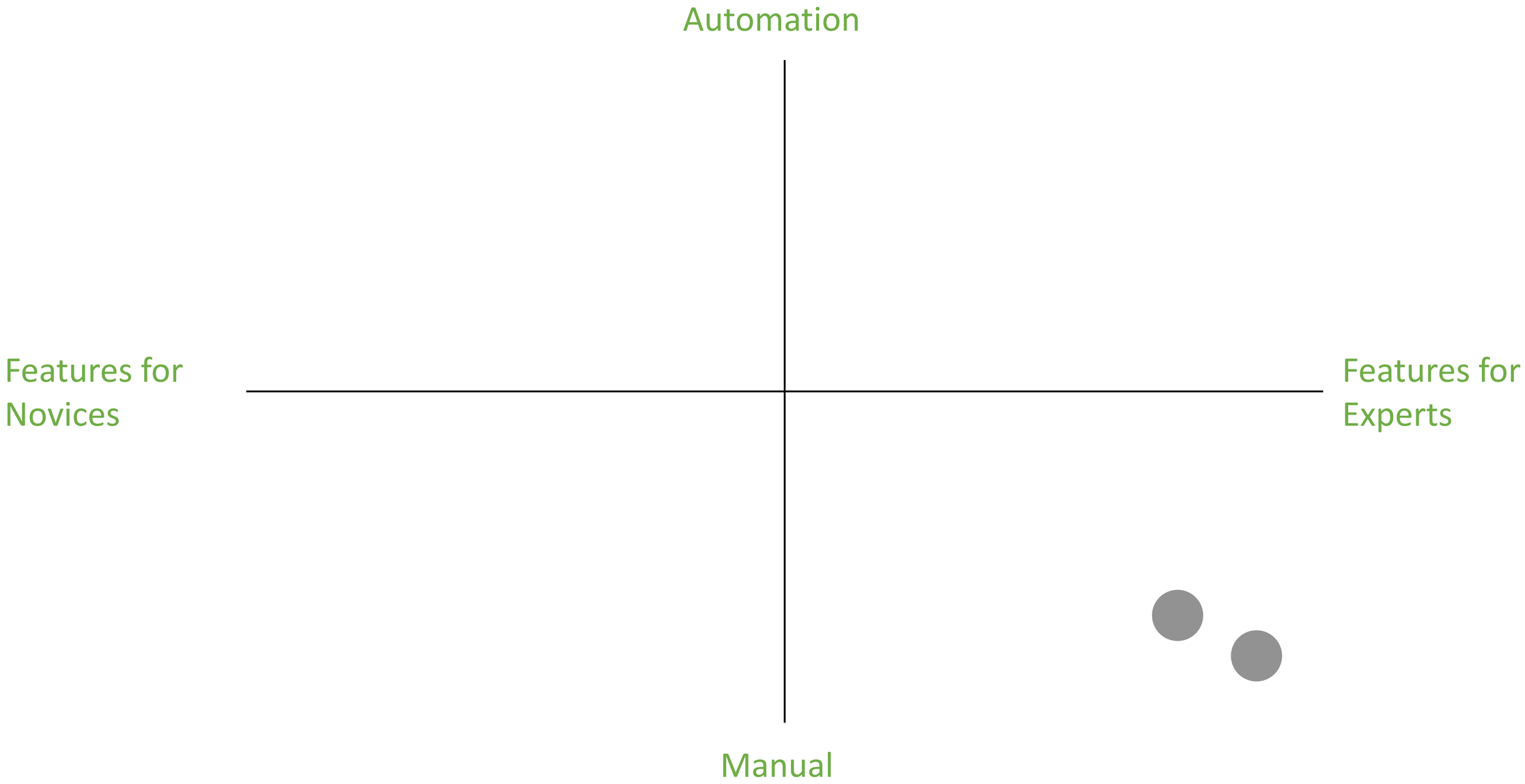
n^2

Users can also take advantage of advanced features provided by the language, such as the ability for the verifier to provide random values during the execution of a protocol, in order to further optimize its performance

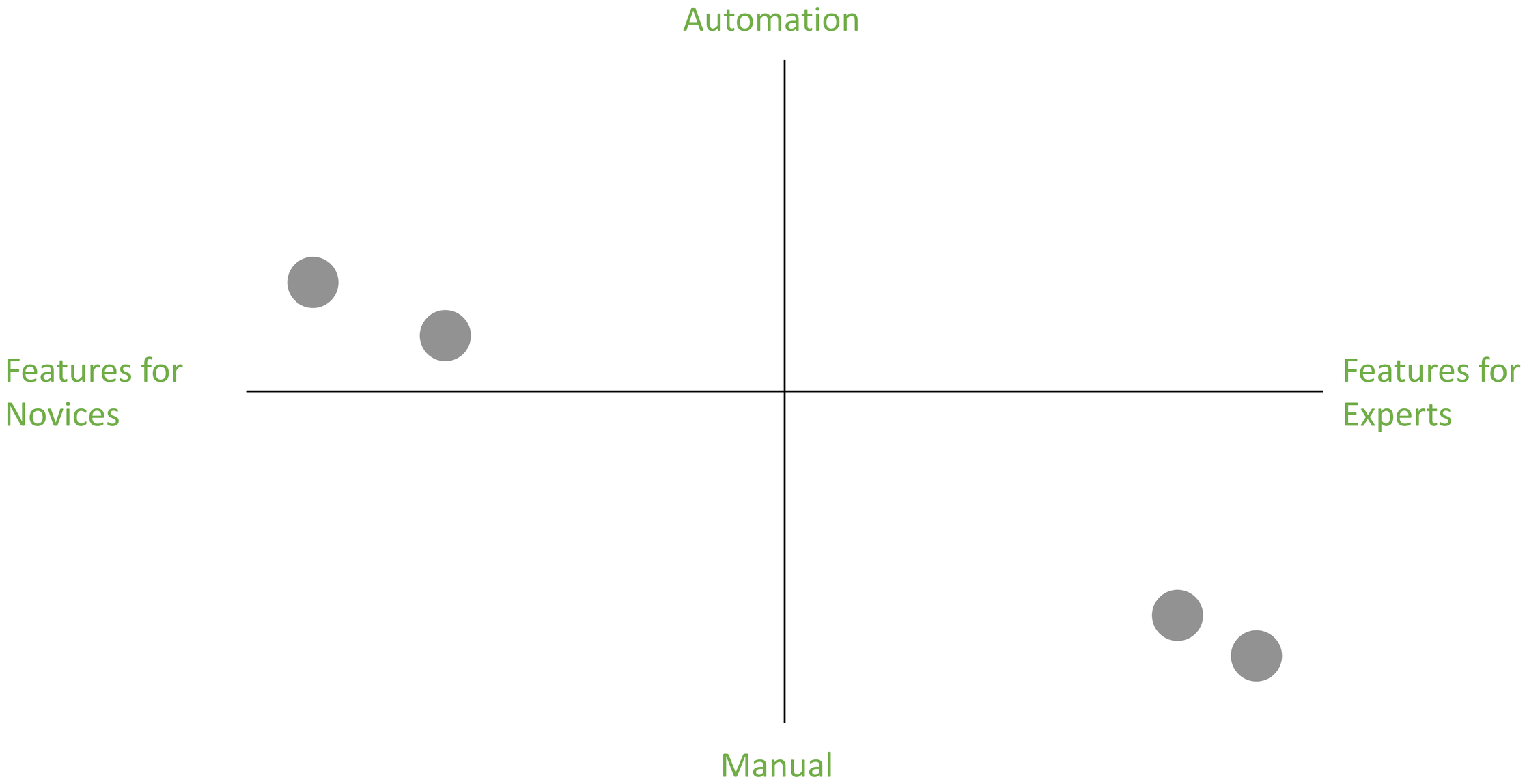
Programmers and Zero-Knowledge Proofs



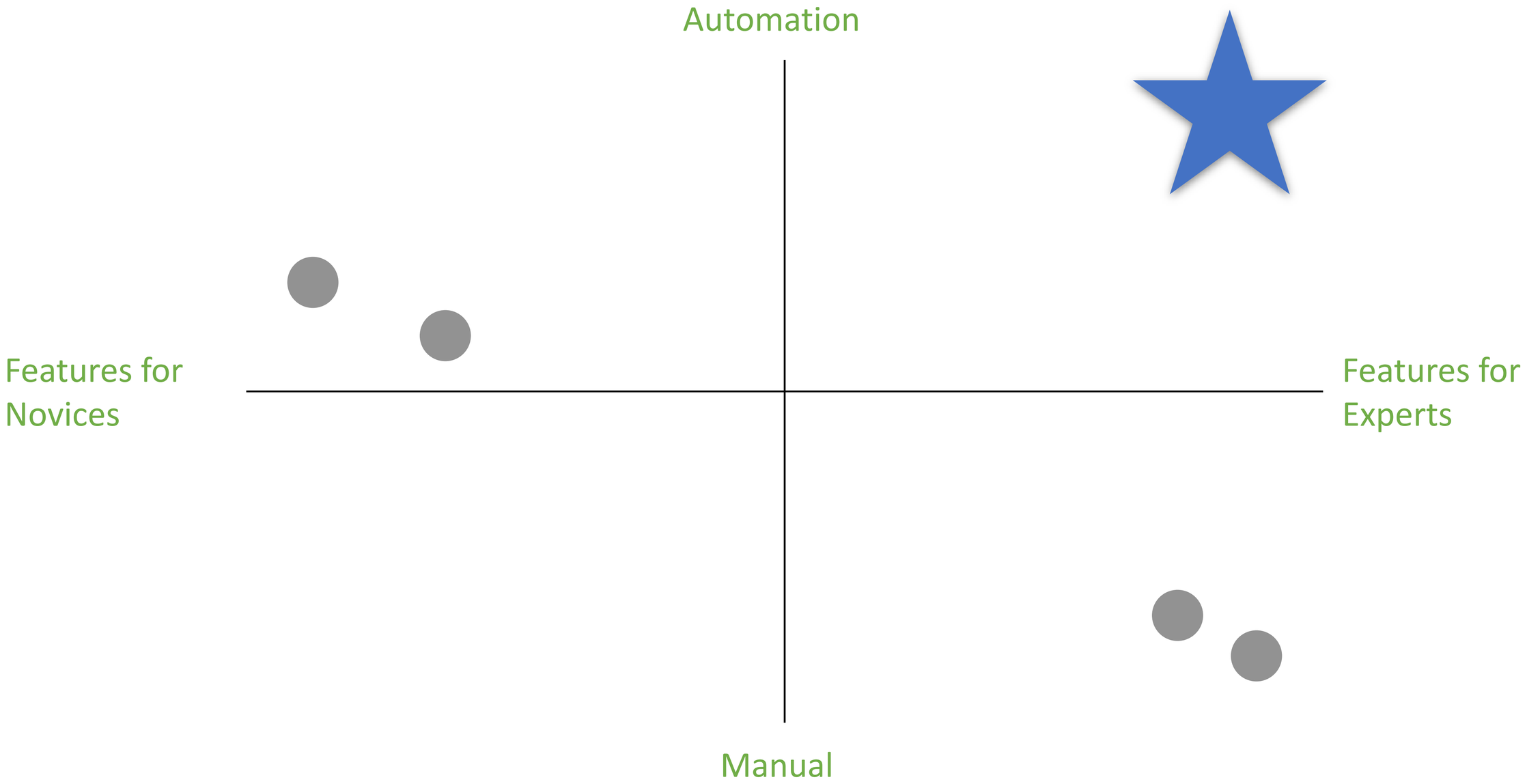
Programmers and Zero-Knowledge Proofs



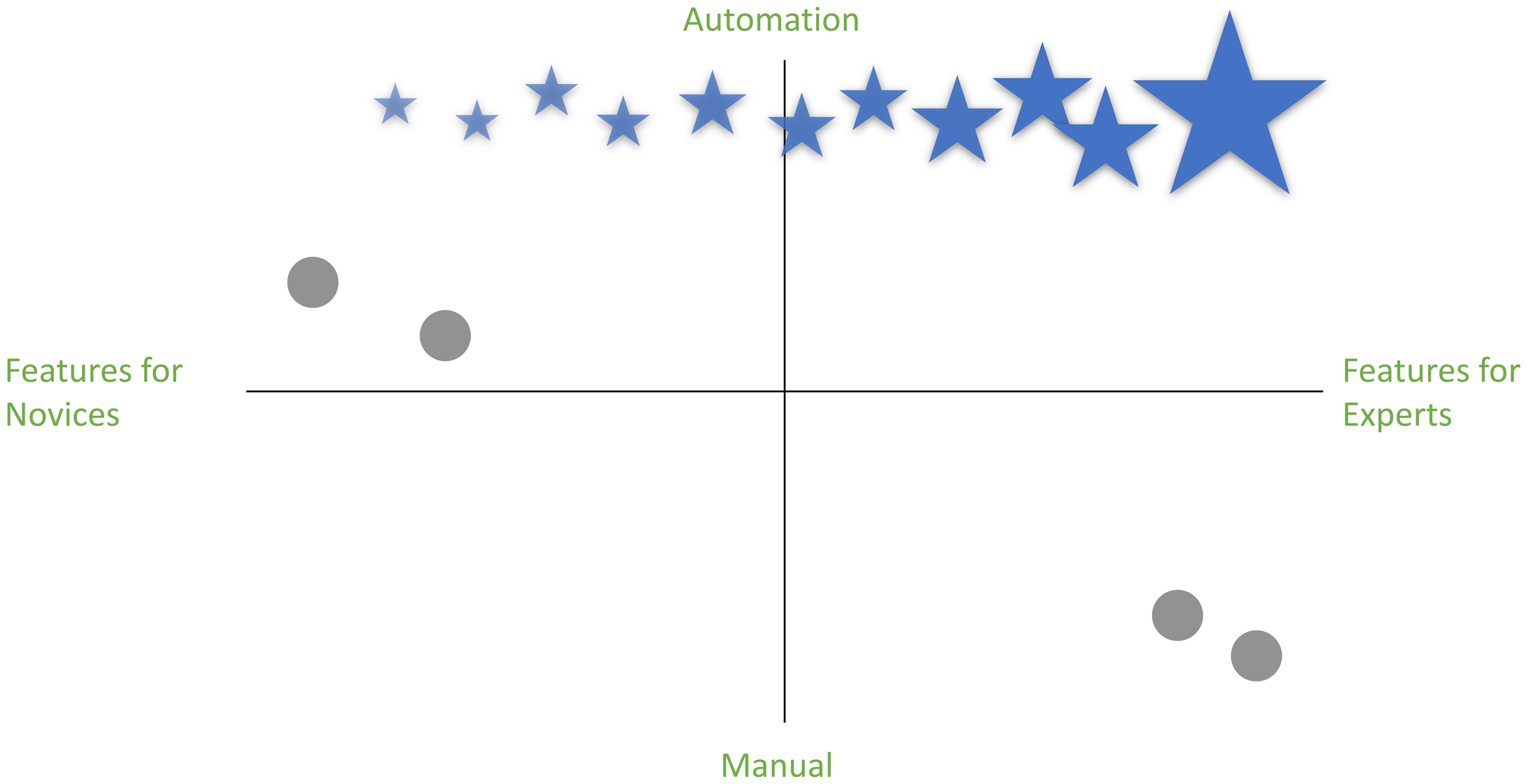
Programmers and Zero-Knowledge Proofs

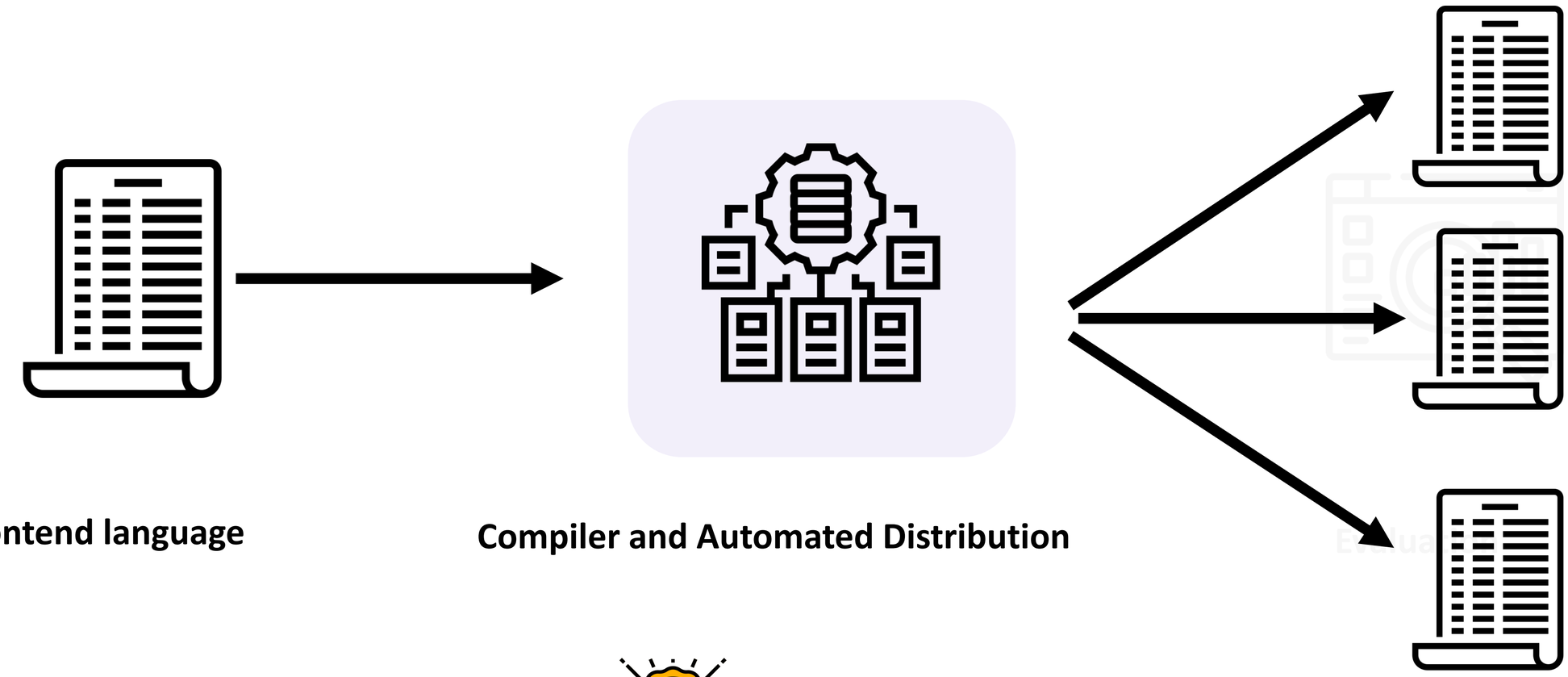


Programmers and Zero-Knowledge Proofs



Programmers and Zero-Knowledge Proofs





Frontend language

Compiler and Automated Distribution

Provably Correct; Guaranteed Secure; Near-Optimal.

Distribute the Zero-Knowledge Proofs

Scaling out is always better than scaling up!

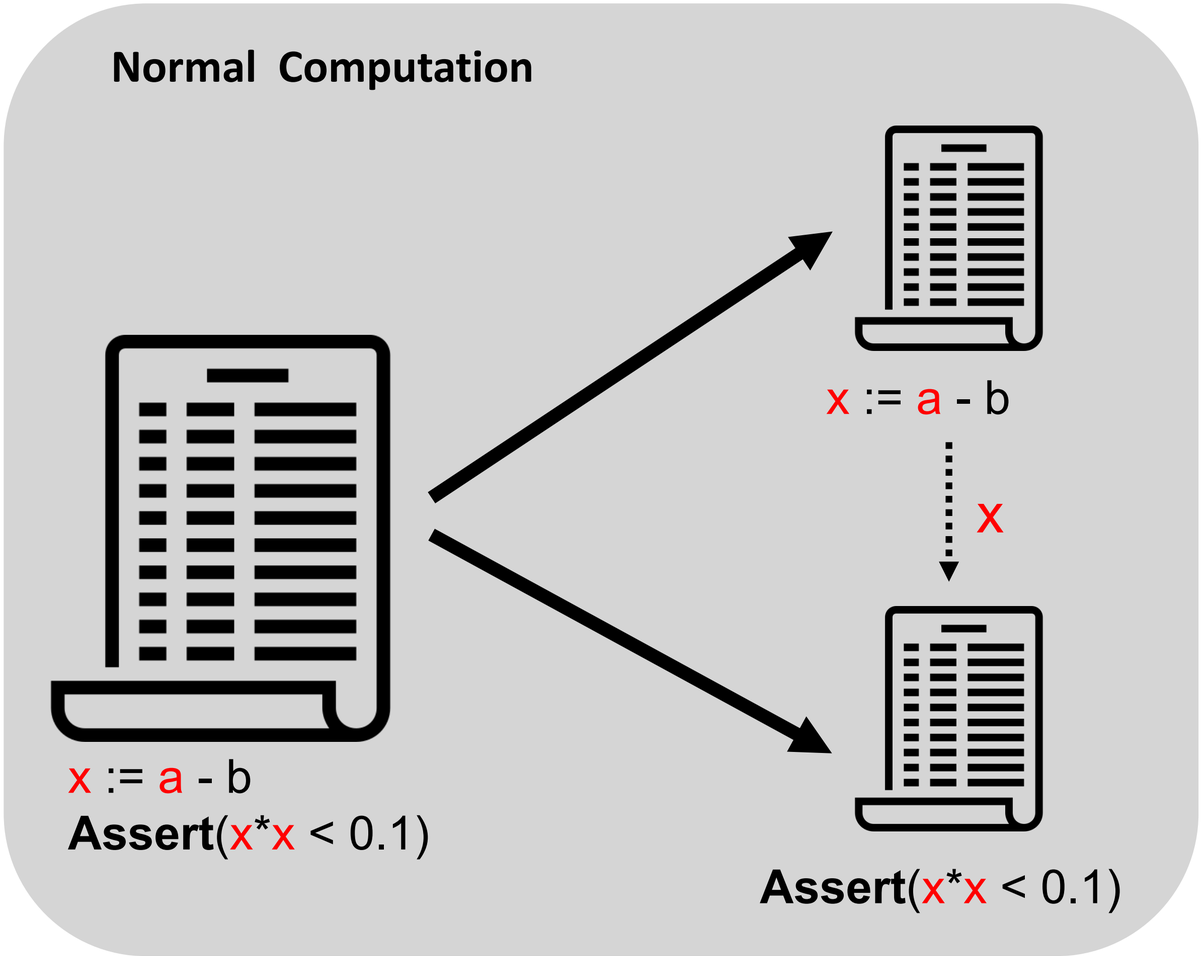
The provers need **significant computation resource.**

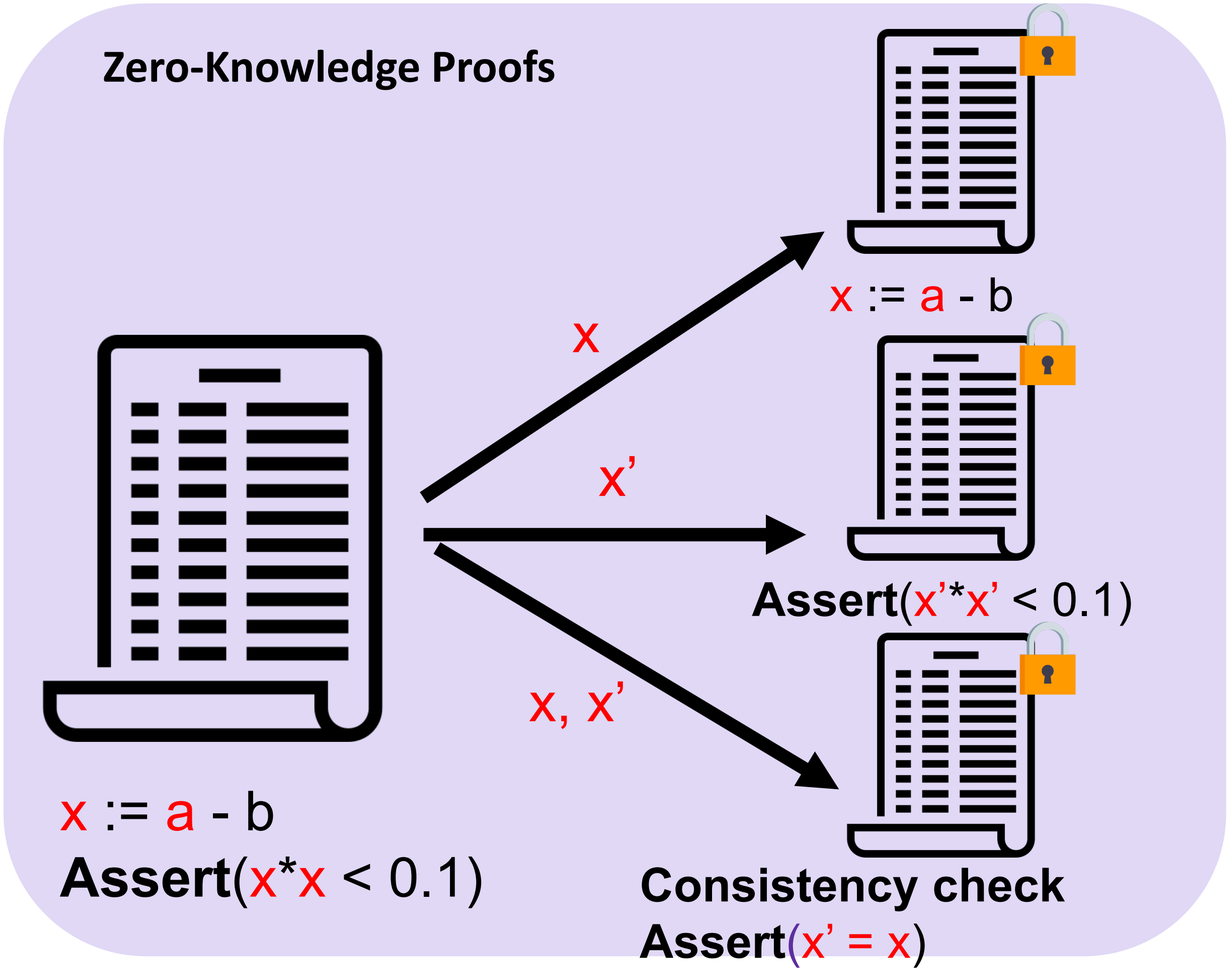
zkSNARK require massive hardware resources in practice

- **Terabytes of RAM** are required to generate real-world proofs!

Distribute the Zero-Knowledge Proofs

ZKPs are exceptionally well-suited for distribution.



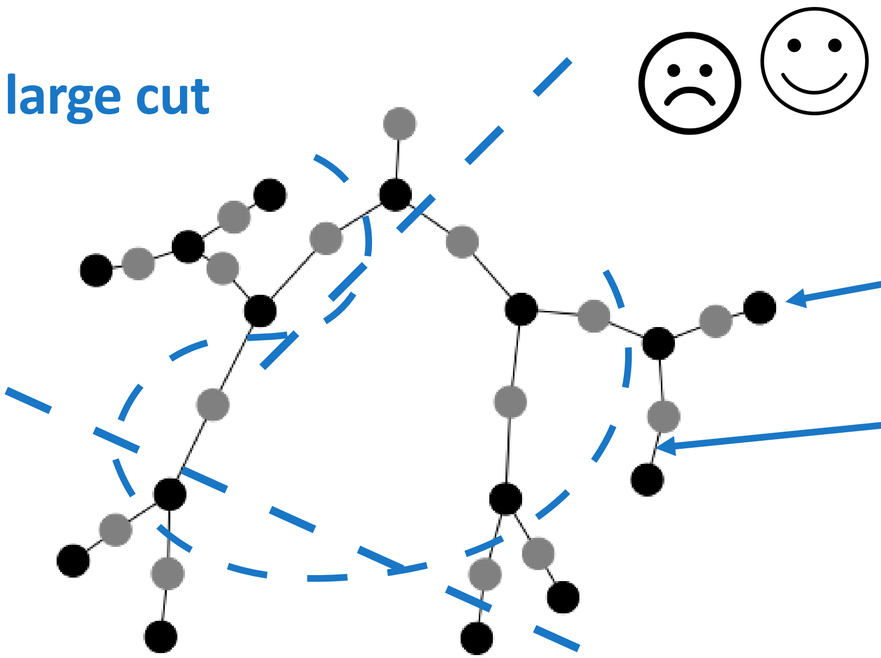


Data dependency ⇒ Block Waiting

Precomputing dependency ⇒ No Block Waiting

Partition: Find the Optimal Distribution

Take dependency graphs as inputs, and output the optimal distribution scheme



large cut

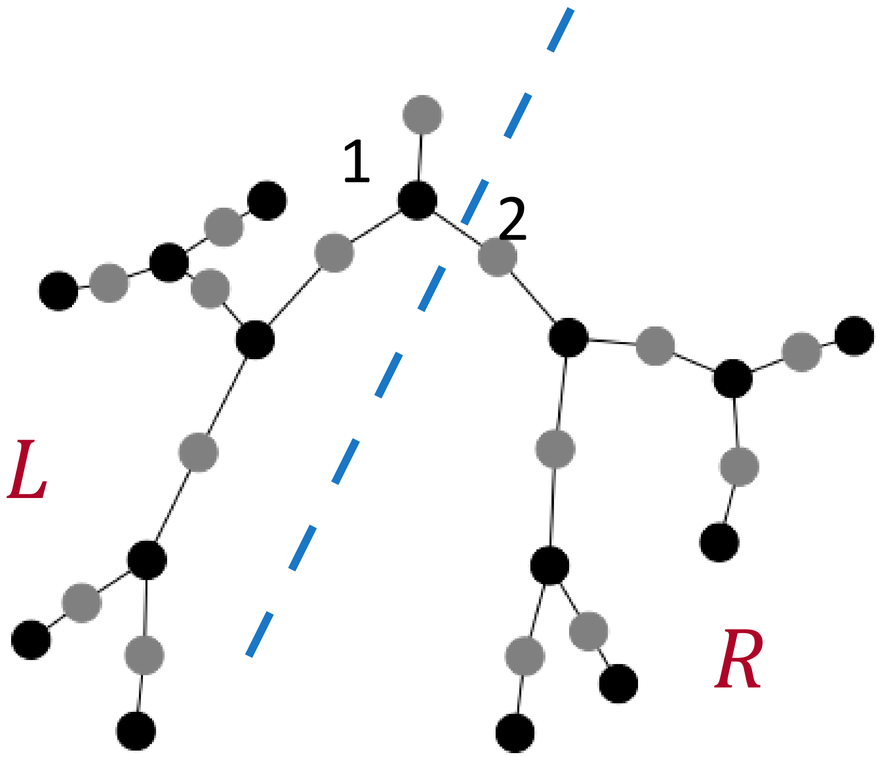
small cut, balanced workload

unbalanced partition

- Nodes represent each instruction
  - Annotate with the cost of each instruction P_i
- Edges represents the data dependency
  - Annotate with the cost of consistency check $C_{i,j}$

Partition: Find the Optimal Distribution

Model optimal distribution search as Integer Linear Programming (ILP) problem



- Nodes represent each instruction
  - Annotate with the cost of each instruction P_i
  - $X_{it} \in \{0,1\}$ indicates if Node i will be in the tth partition
- Edges represents the data dependency
  - Annotate with the cost of consistency check $C_{i,j}$
  - $Y_{ij} \in \{0,1\}$ indicates if a cut is made at the edge (i,j)

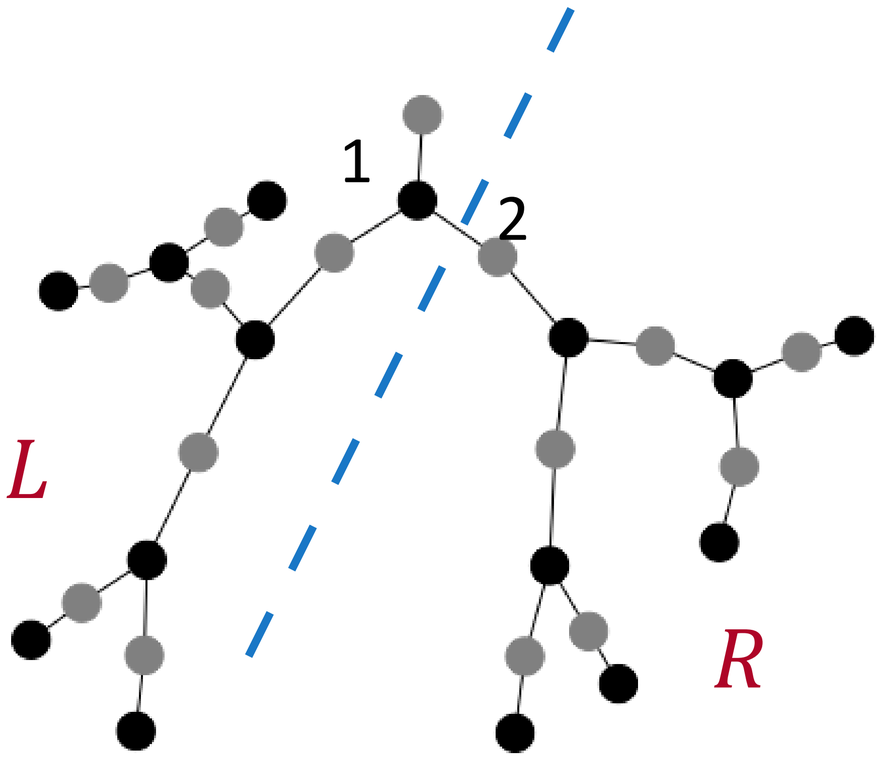
$Y_{1,2} = 1$, and $Y_{i,j} = 0$ for other $(i,j) \in (R \cup L)^2$

$X_{r,0} = 1$ for $r \in R$, $X_{\ell,0} = 0$ for $\ell \in L$

$X_{r,0} = 0$ for $r \in R$, $X_{\ell,0} = 1$ for $\ell \in L$

Partition: Find the Optimal Distribution

Model Optimal distribution Search as Integer Linear Programming problem



cost = $C_{1,2} + max\{\sum P_r, \sum P_\ell\}$

- Nodes represent each instruction
  - Annotate with the cost of each instruction P_i
  - $X_{it} \in \{0,1\}$ indicates if Node i will be in the tth partition
- Edges represents the data dependency
  - Annotate with the cost of consistency check $C_{i,j}$
  - $Y_{ij} \in \{0,1\}$ indicates if a cute is made at the edge (i,j)

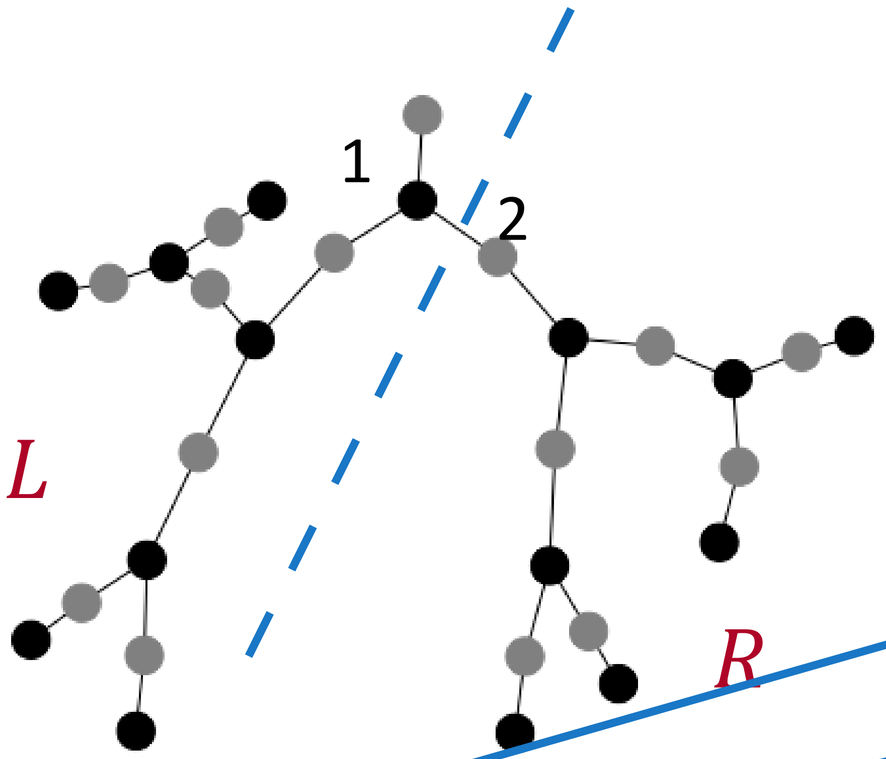
$Y_{1,2} = 1,$ and $Y_{i,j} = 0$ for other $(i,j) \in (R \cup L)^2$

$X_{r,0} = 1$ for $r \in R,\ \ X_{\ell,0} = 0$ for $\ell \in L$

$X_{r,0} = 0$ for $r \in R,\ \ X_{\ell,0} = 1$ for $\ell \in L$

Partition: Find the Optimal Distribution

Model Optimal distribution Search as Integer Linear Programming problem



cost = $C_{1,2} + max\{\sum P_r, \sum P_\ell\}$

- Nodes represent each instruction
  - Annotate with the cost of each instruction P_i
  - $X_{it} \in \{0,1\}$ indicates if Node i will be in the tth partition
- Edges represent the data dependency
  - A[illegible] of consistency check $C_{i,j}$
  - $Y_{i,j} \in \{0,1\}$ indicates if a cute is made at the edge (i,j)

Cost of consistency check

The largest cost of subprovers

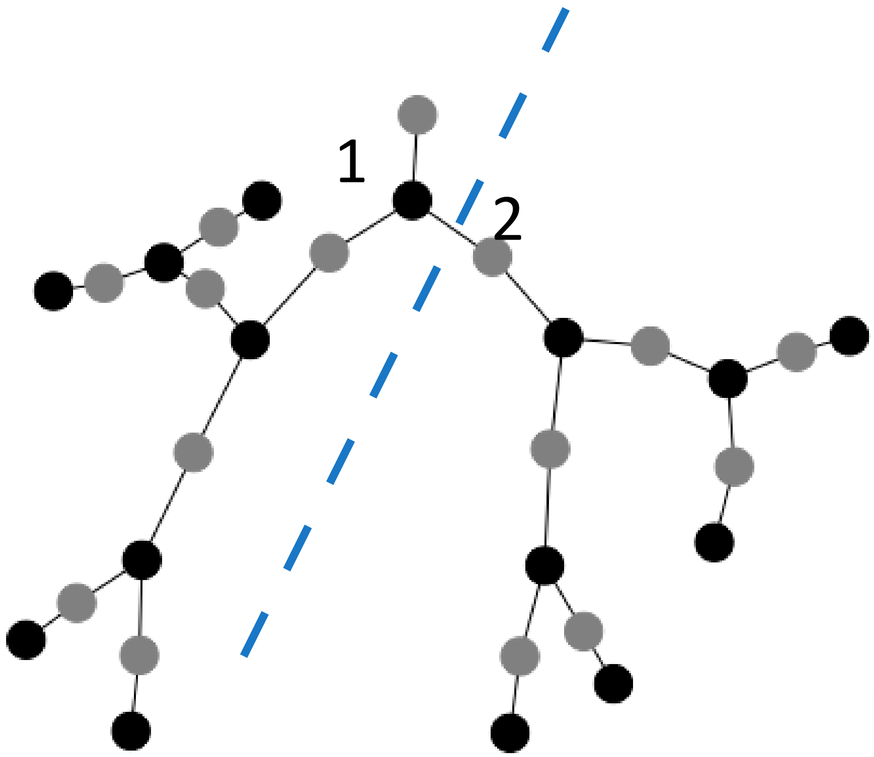
$Y_{1,2} = 1$, and $Y_{i,j} = 0$ for other $(i,j) \in (R \cup L)^2$

$X_{r,0} = 1$ for $r \in R$, $X_{\ell,0} = 0$ for $\ell \in L$

$X_{r,0} = 0$ for $r \in R$, $X_{\ell,0} = 1$ for $\ell \in L$

Partition: Find the Optimal Distribution

Model Optimal distribution Search as Integer Linear Programming problem



- Nodes represent each instruction
  - Annotate with the cost of each instruction P_i
  - $X_{it} \in \{0,1\}$ indicates if Node i will be in the tth partition
- Edges represents the data dependency
  - Annotate with the cost of consistency check $C_{i,j}$
  - $Y_{i,j} \in \{0,1\}$ indicates if a cute is made at the edge (i, j)

Integer Linear Programming (ILP)

ILP Solver

objective: $\sum C_{i,j} \cdot Y_{i,j} + \max_t \sum P_i \cdot X_{it}$

s.t. $\sum_t X_{i,t} = 1$

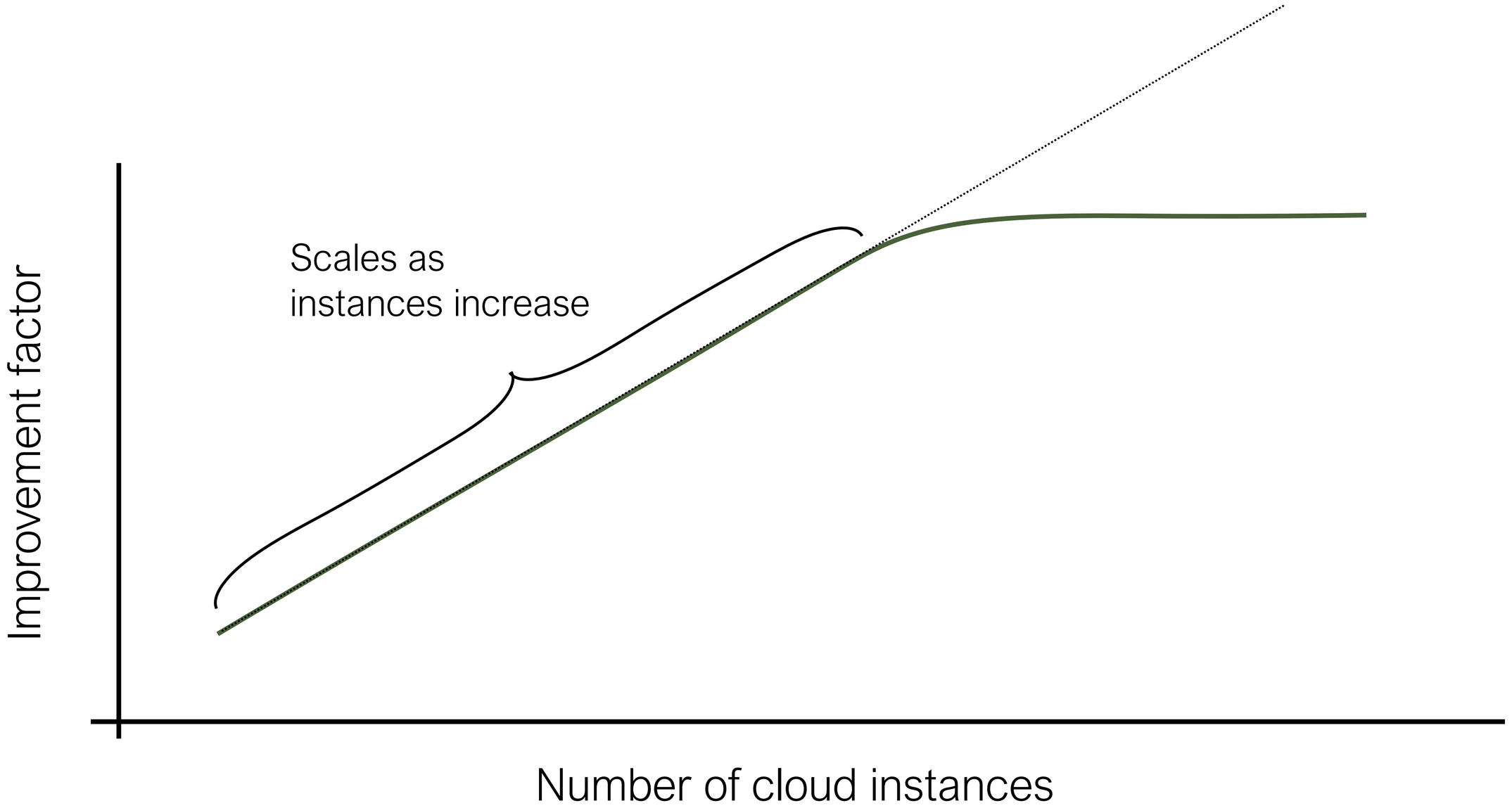
… (other constraints)

$Y_{1,2} = 1$, and $Y_{i,j} = 0$ for other $(i,j) \in (R \cup L)^2$

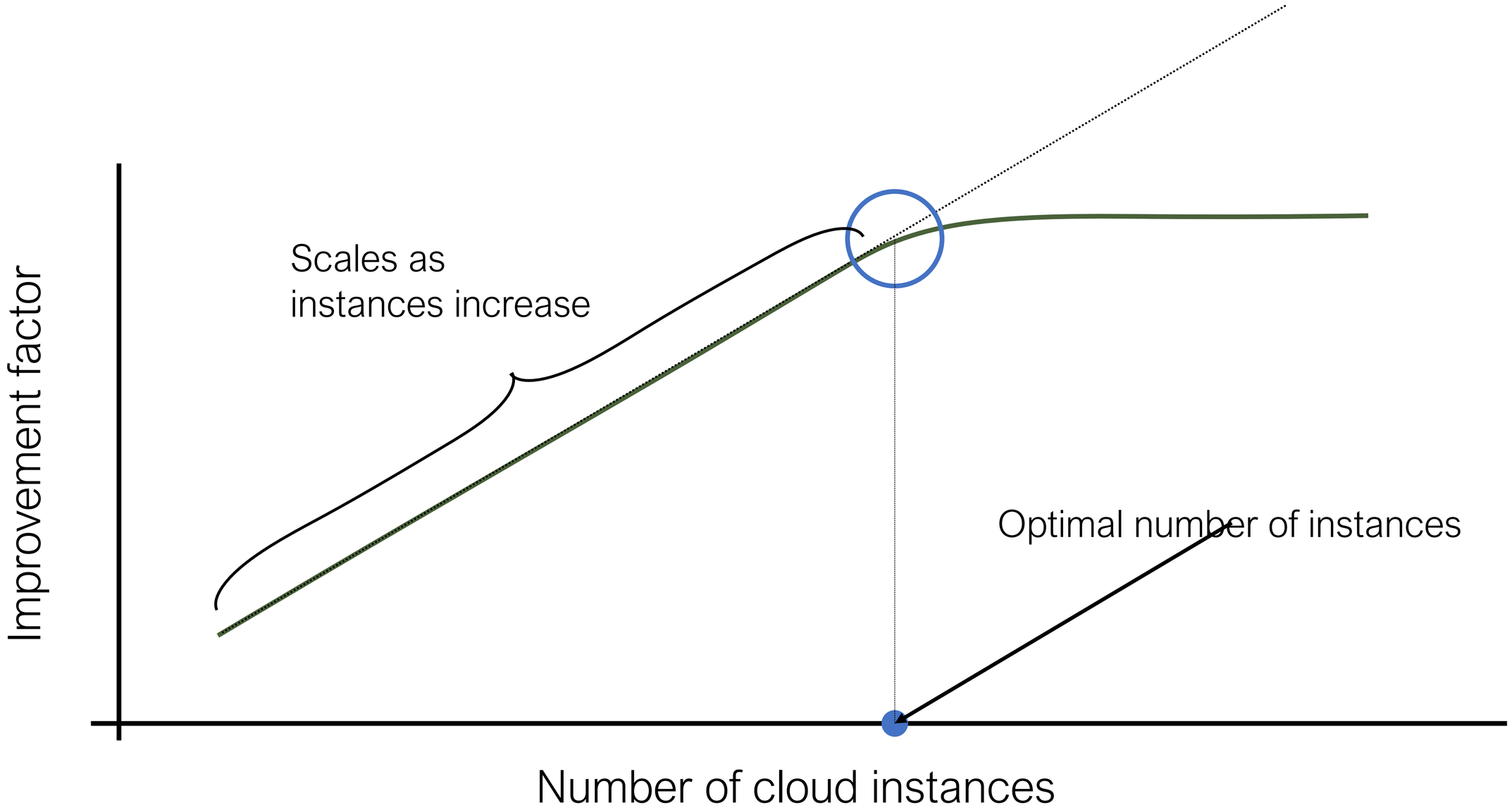
$X_{r,0} = 1$ for $r \in R$, $X_{\ell,0} = 0$ for $\ell \in L$

$X_{r,0} = 0$ for $r \in R$, $X_{\ell,0} = 1$ for $\ell \in L$

Preliminary Results



Preliminary Results



Indicative runtime savings

Our preliminary experiments for Gradient descent and Merkle Tree implementations show that the realized cost savings are often close to the optimal ones.

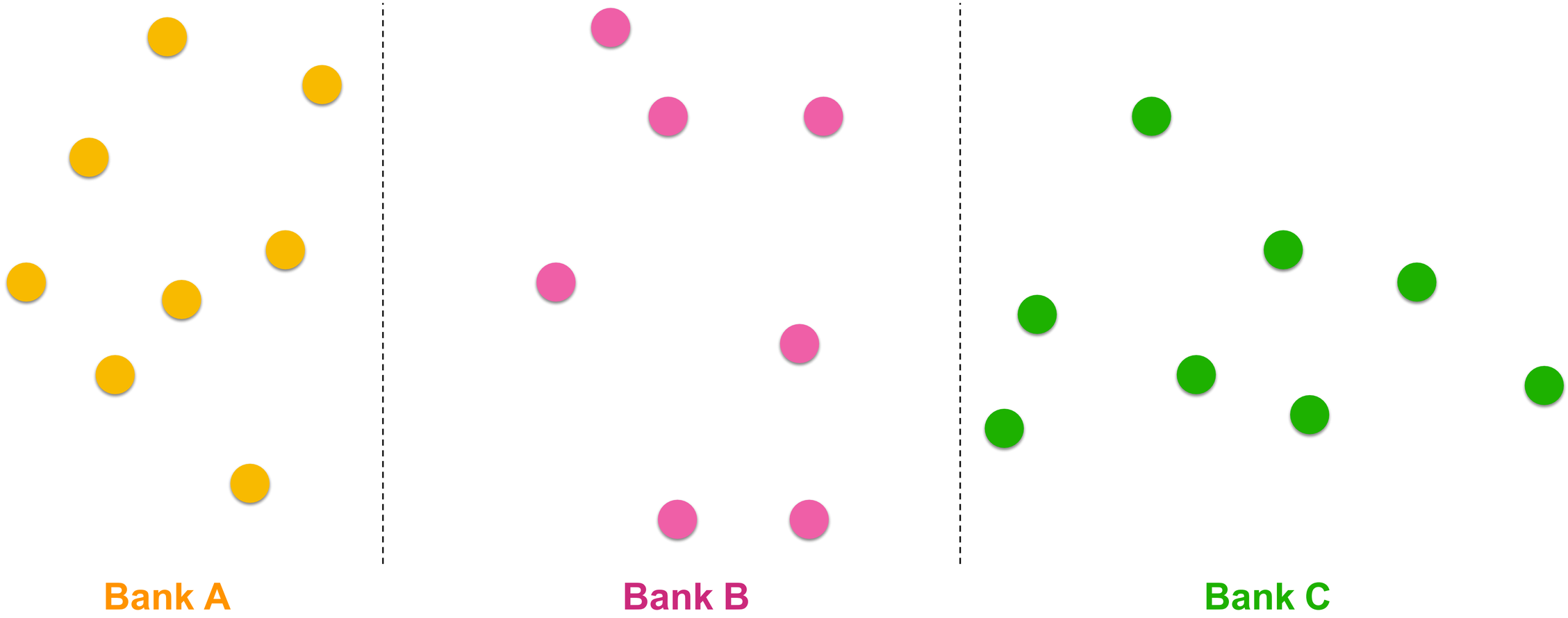
	machines	1	5	10	20	40
Gradient Descent	Runtime	681.55s	147.91s	71.23s	38.06s	21.90s
	Cost ratio	**1**	**4.99**	**9.95**	**19.9**	**39.8**
Merkle Tree	Runtime	35.91s + ~4s	6.76s + ~4s	3.31s + ~4s	2.18s + ~4s	0.93s + ~4s
	Cost ratio	**1**	**4.95**	**9.75**	**19.47**	**38.63**

Privacy-Preserving Automated Reasoning for Financial Crime Detection

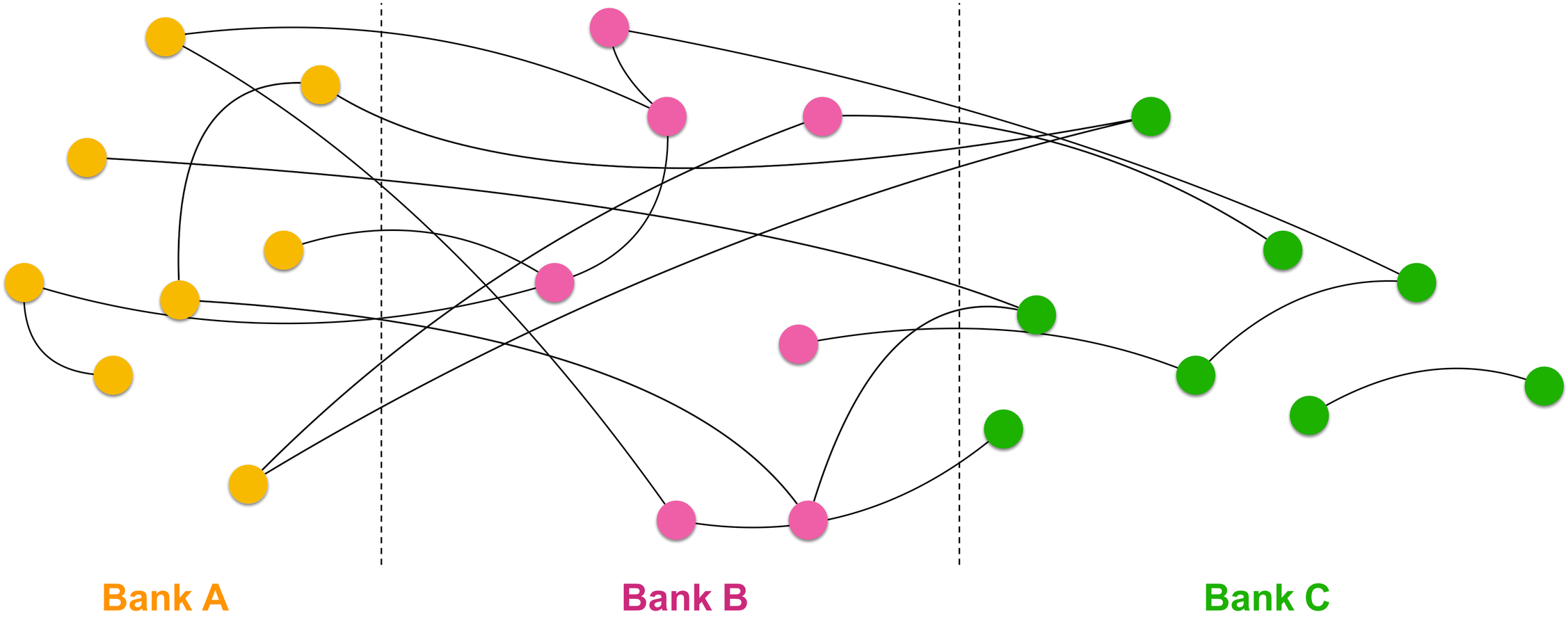
- Modern financial systems process millions of transactions daily.
- Money laundering networks often hide behind:
  - complex transaction chains,
  - cross-border transfers,
  - cyclic transaction patterns.
- Financial institutions must detect suspicious behavior while preserving customer privacy and complying with regulations.

Use **privacy-preserving automated reasoning** to detect suspicious financial structures without exposing sensitive financial data.

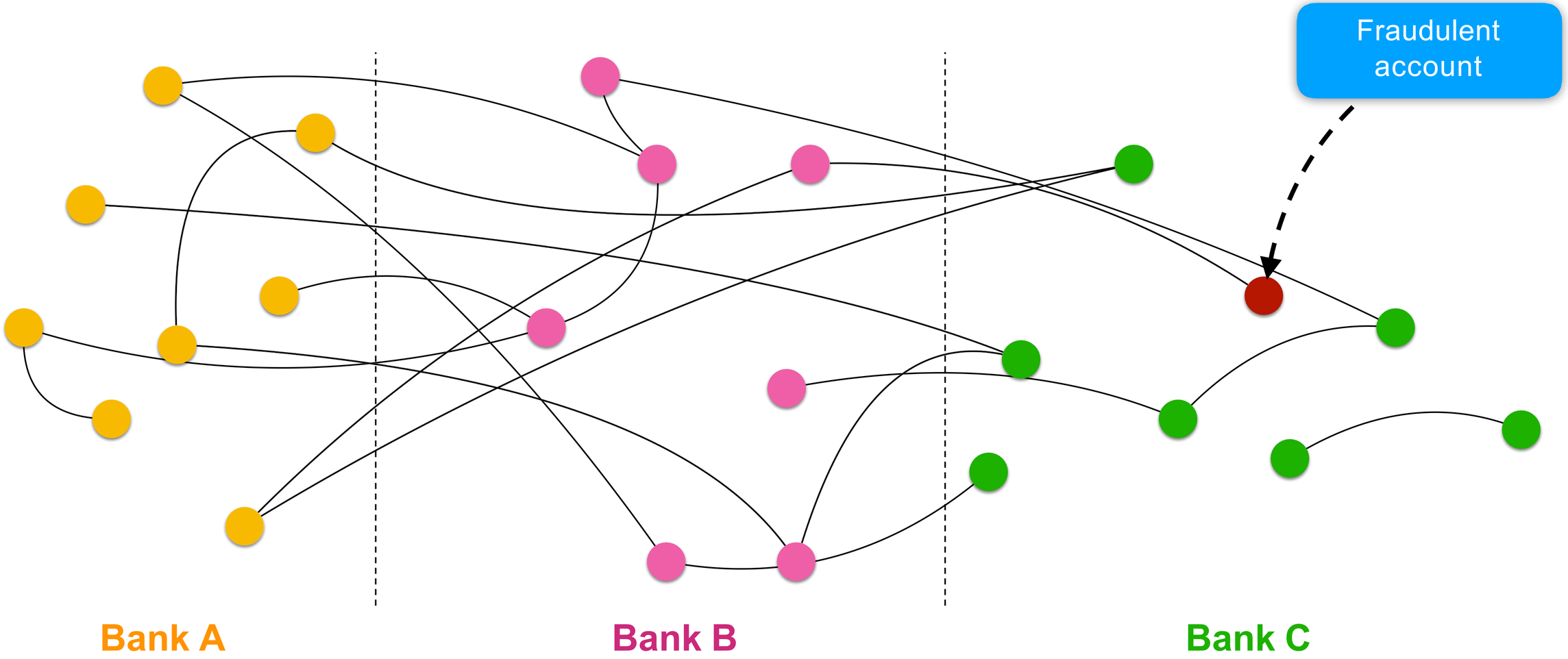
Privacy-Preserving Constraint Solving



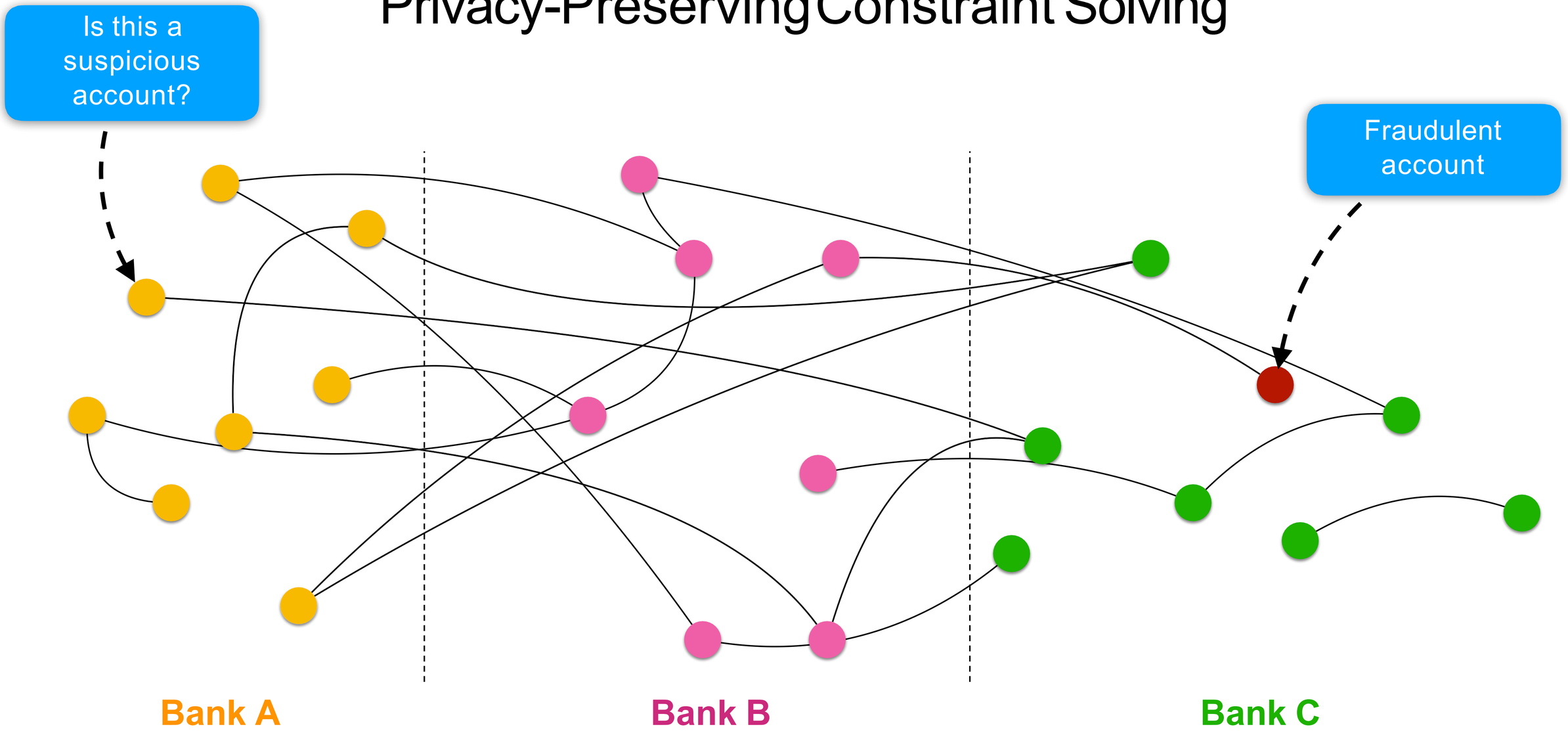
Privacy-Preserving Constraint Solving



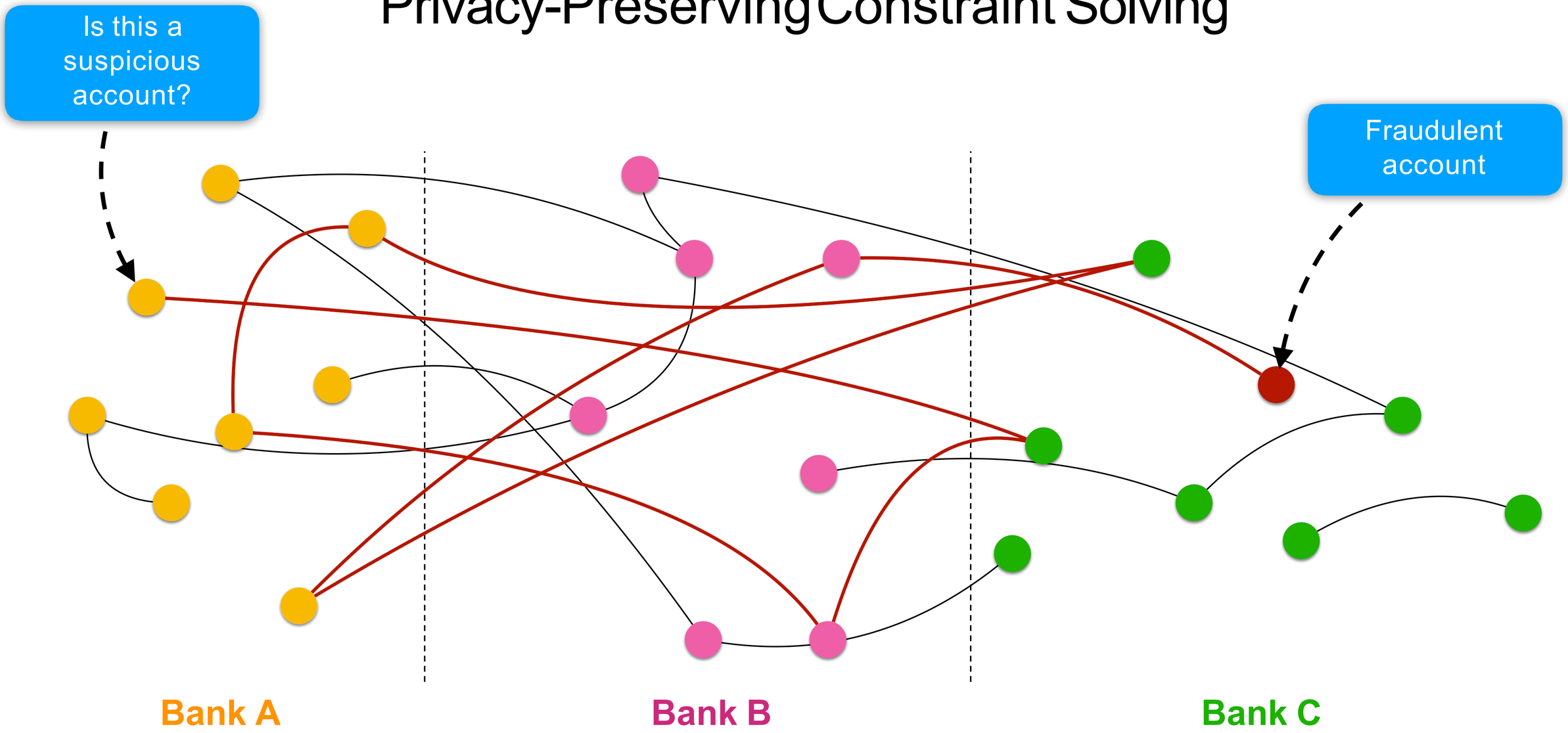
Privacy-Preserving Constraint Solving



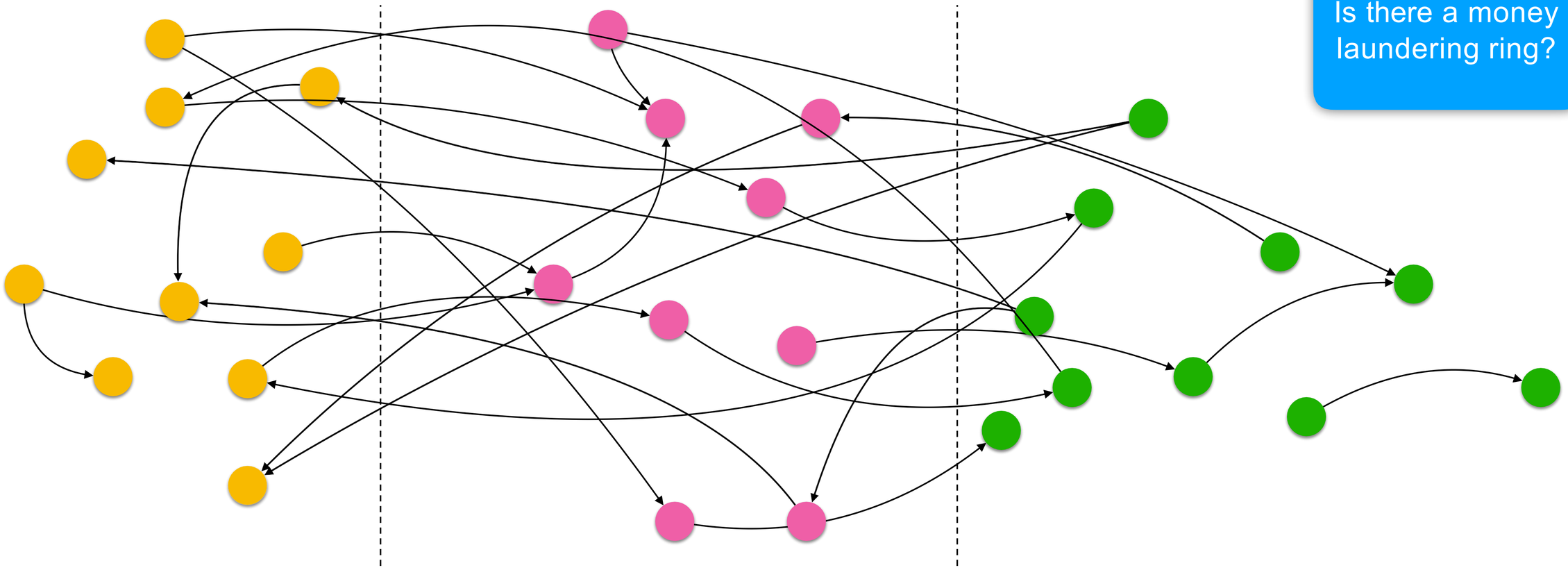
Privacy-Preserving Constraint Solving



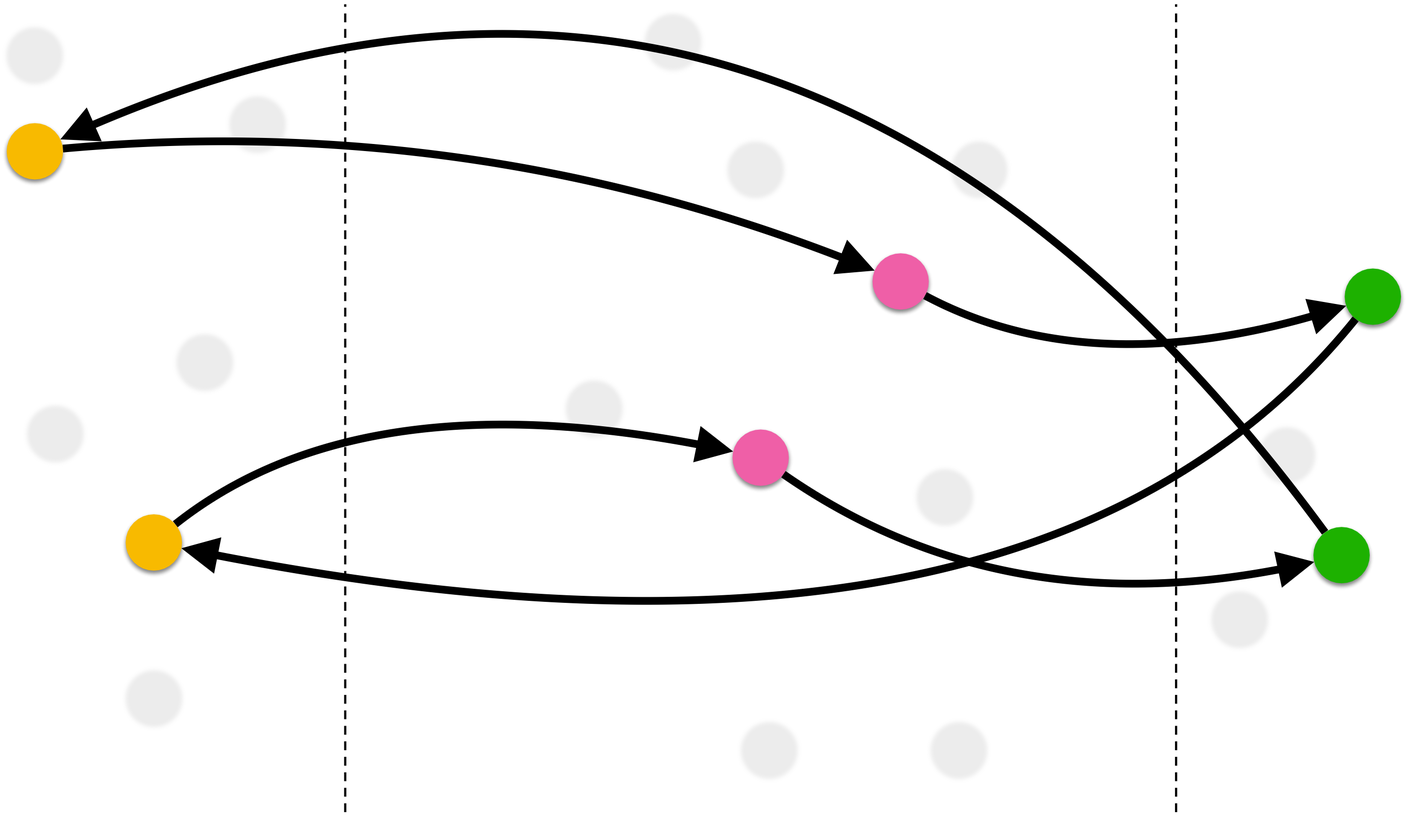
Privacy-Preserving Constraint Solving



Privacy-Preserving Constraint Solving



Privacy-Preserving Constraint Solving

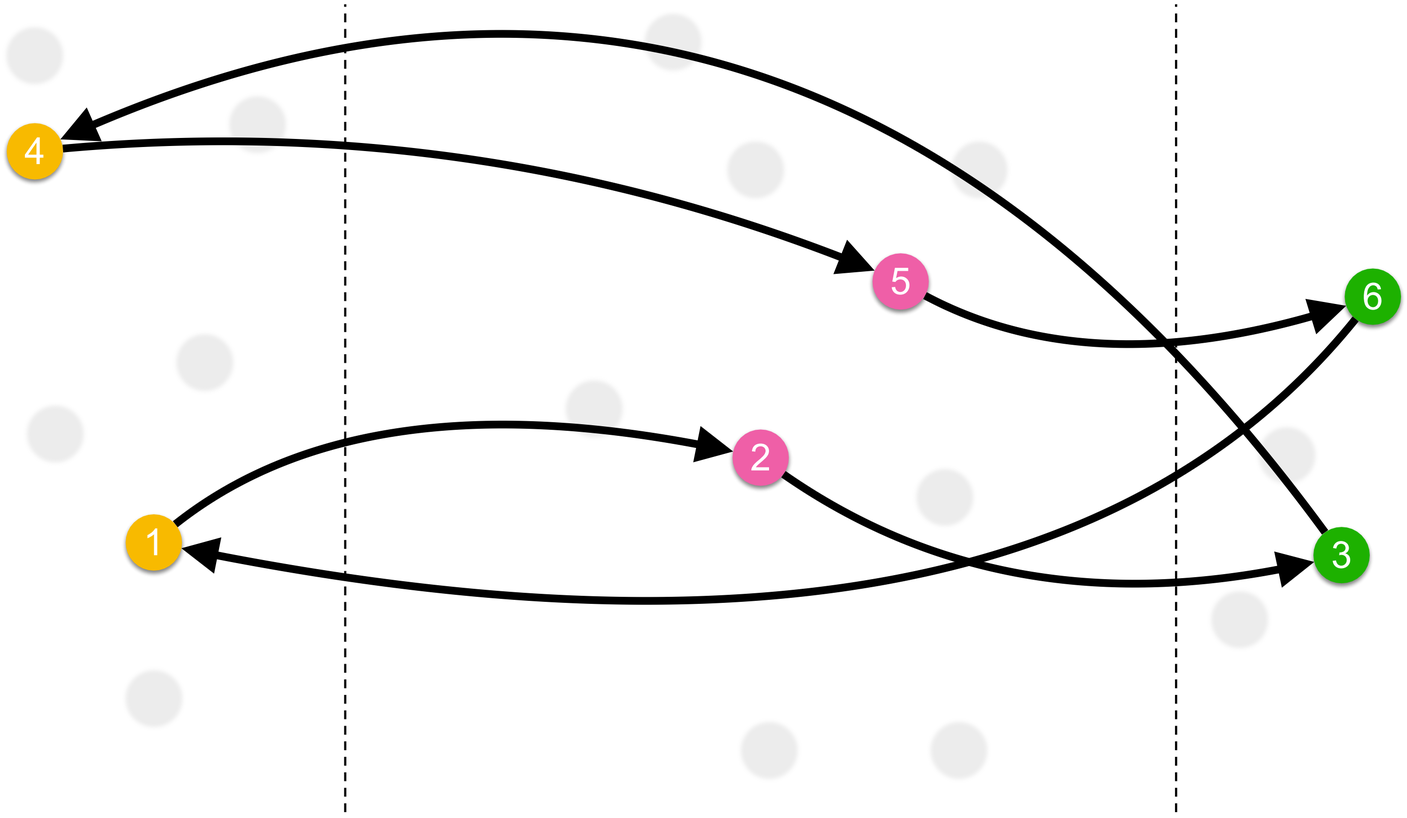


Is there a money laundering ring?

Bank A **Bank B** **Bank C**

Privacy-Preserving Constraint Solving

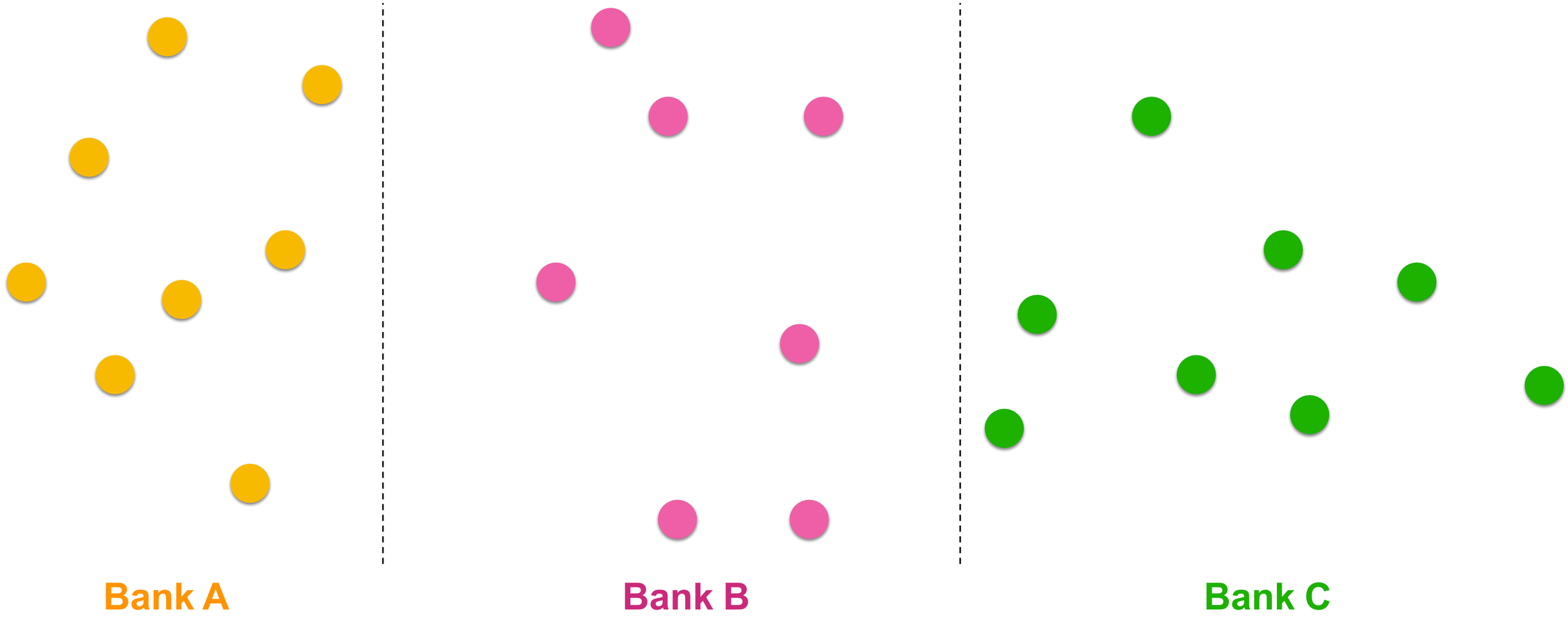
Is there a money laundering ring?



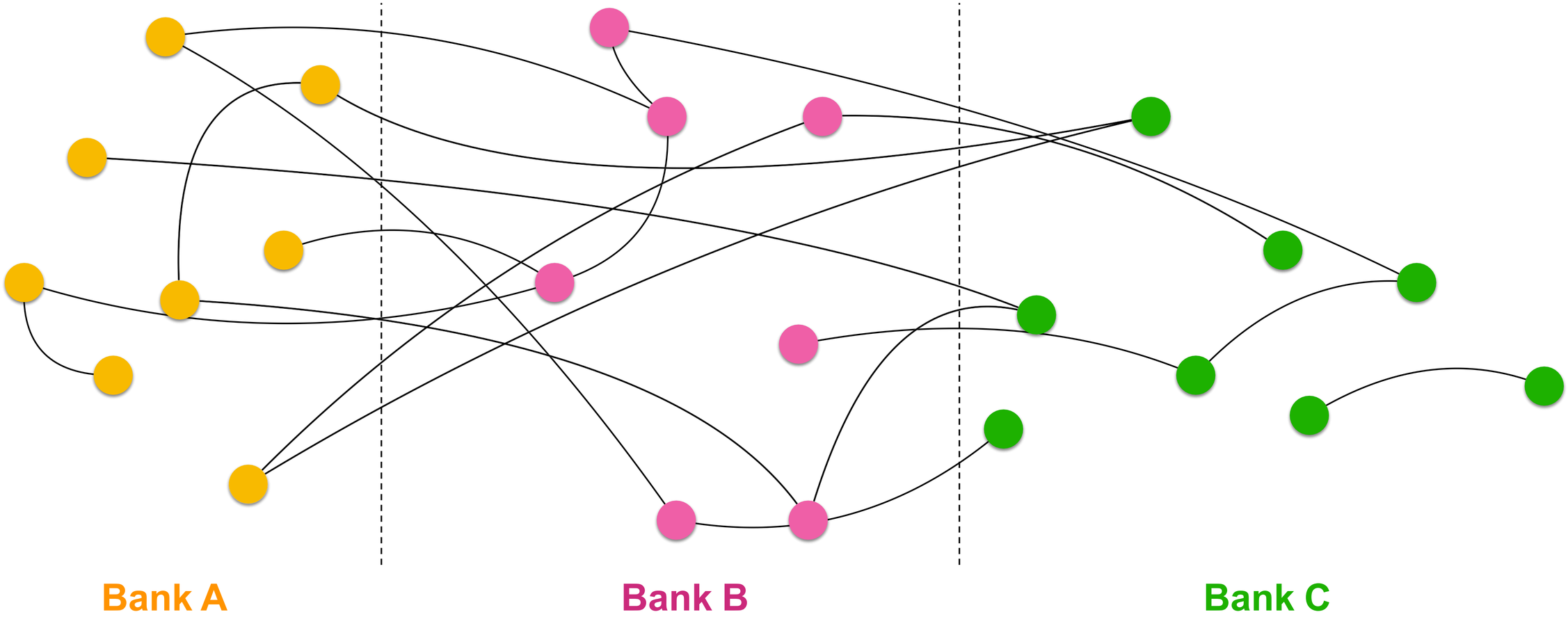
Bank A **Bank B** **Bank C**

1 2 3 4 5 6

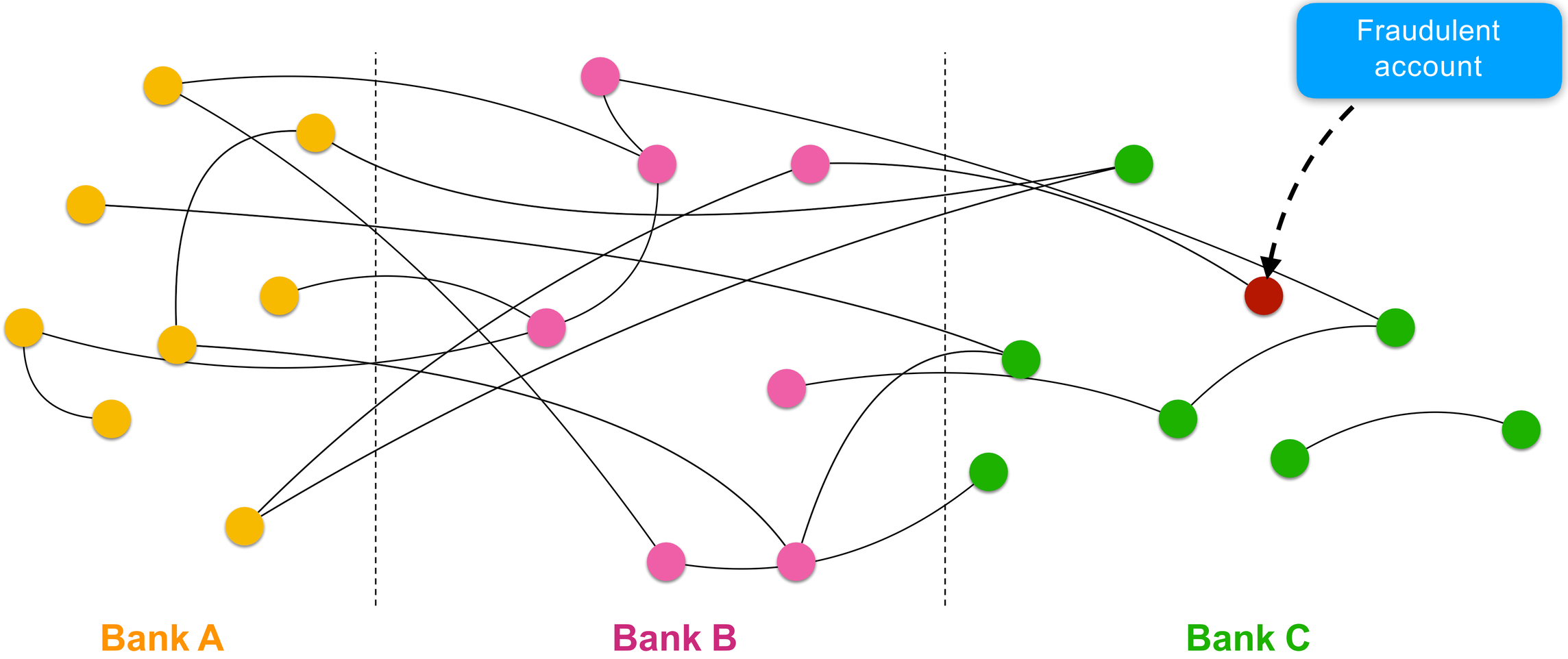
Privacy-Preserving Constraint Solving



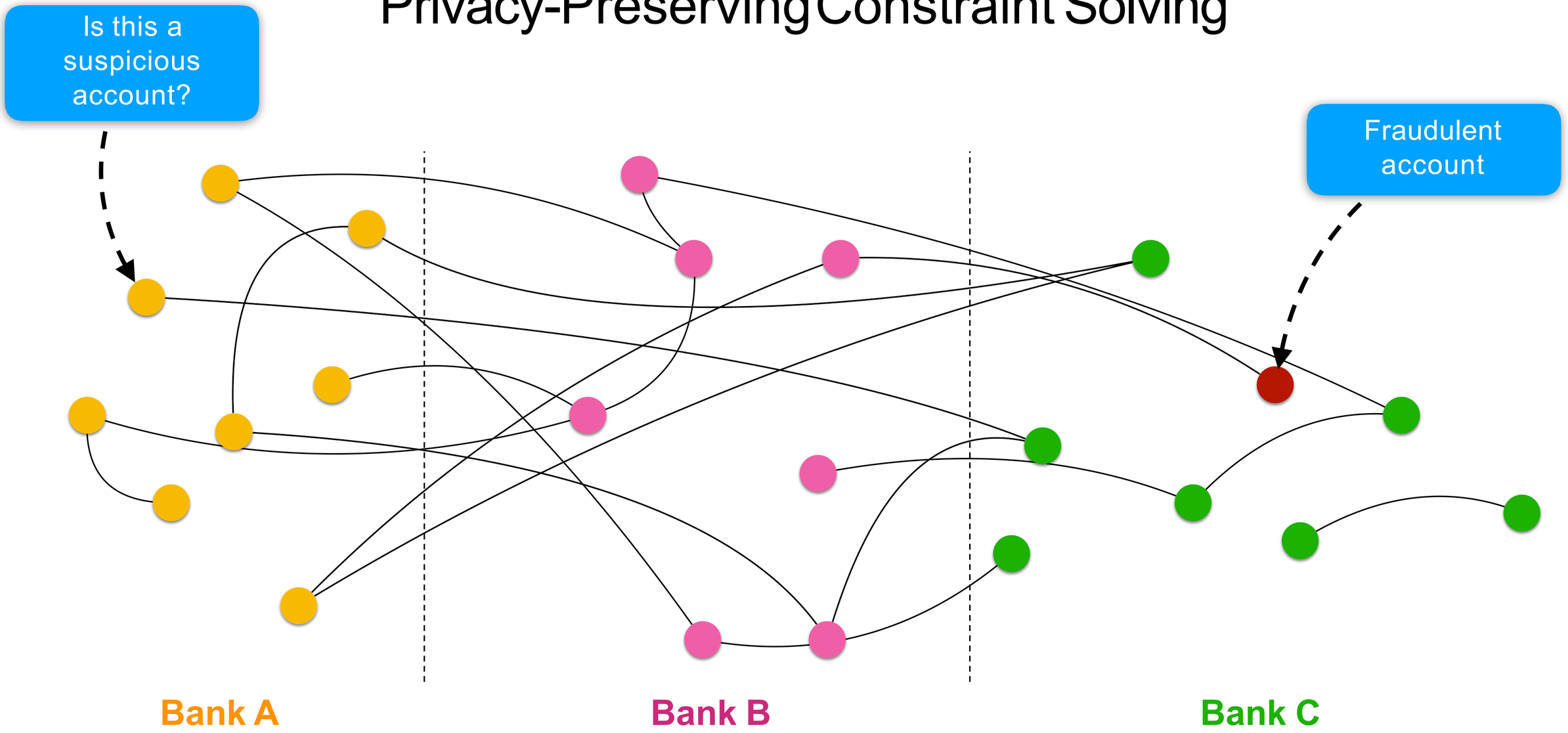
Privacy-Preserving Constraint Solving



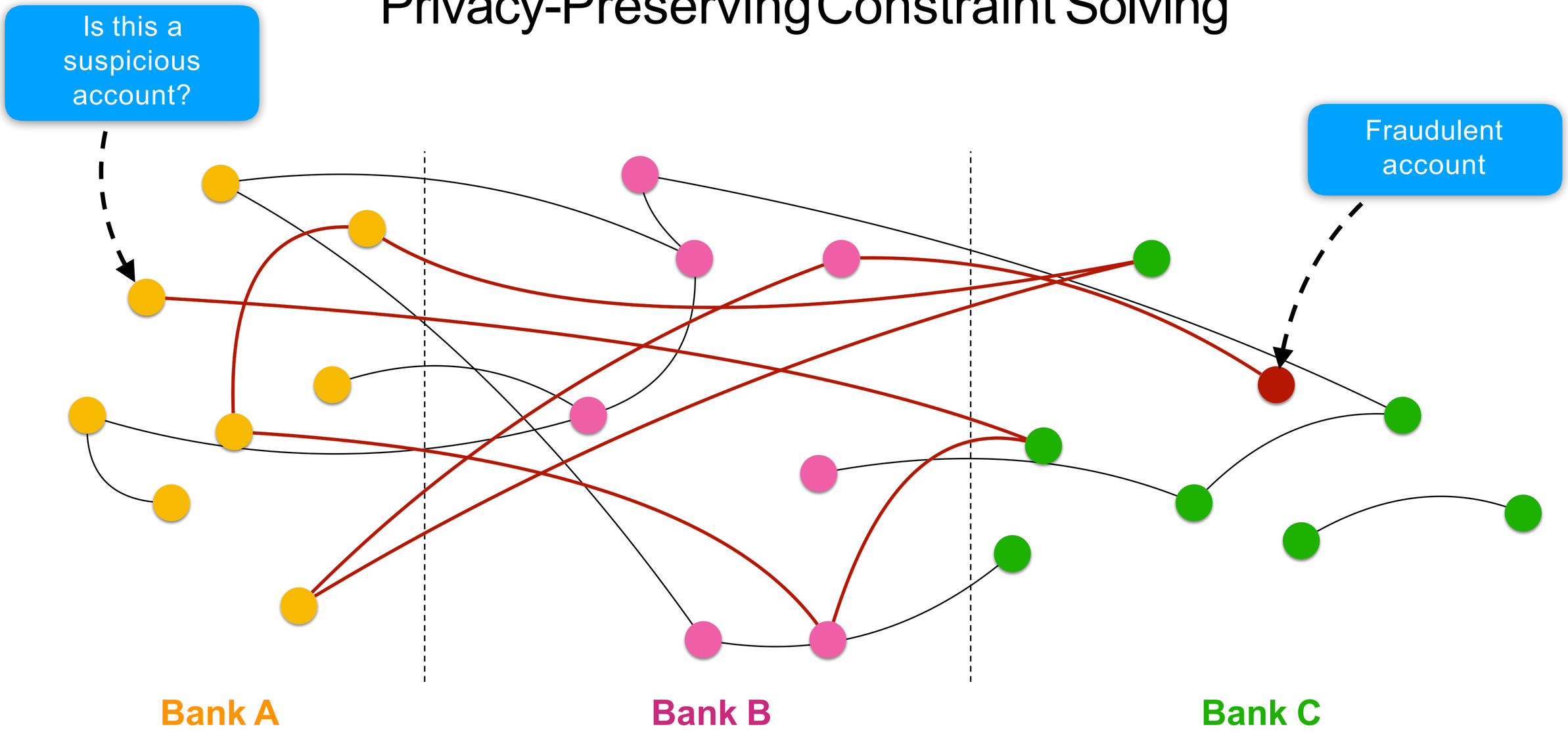
Privacy-Preserving Constraint Solving



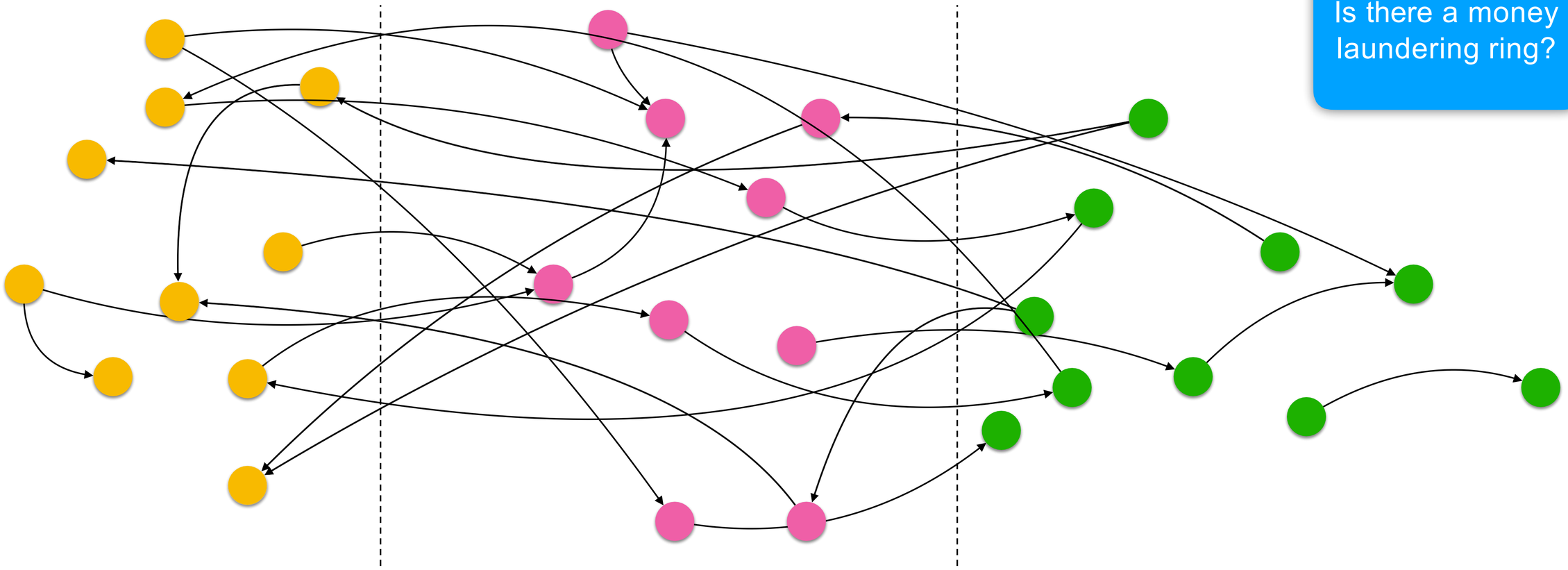
Privacy-Preserving Constraint Solving



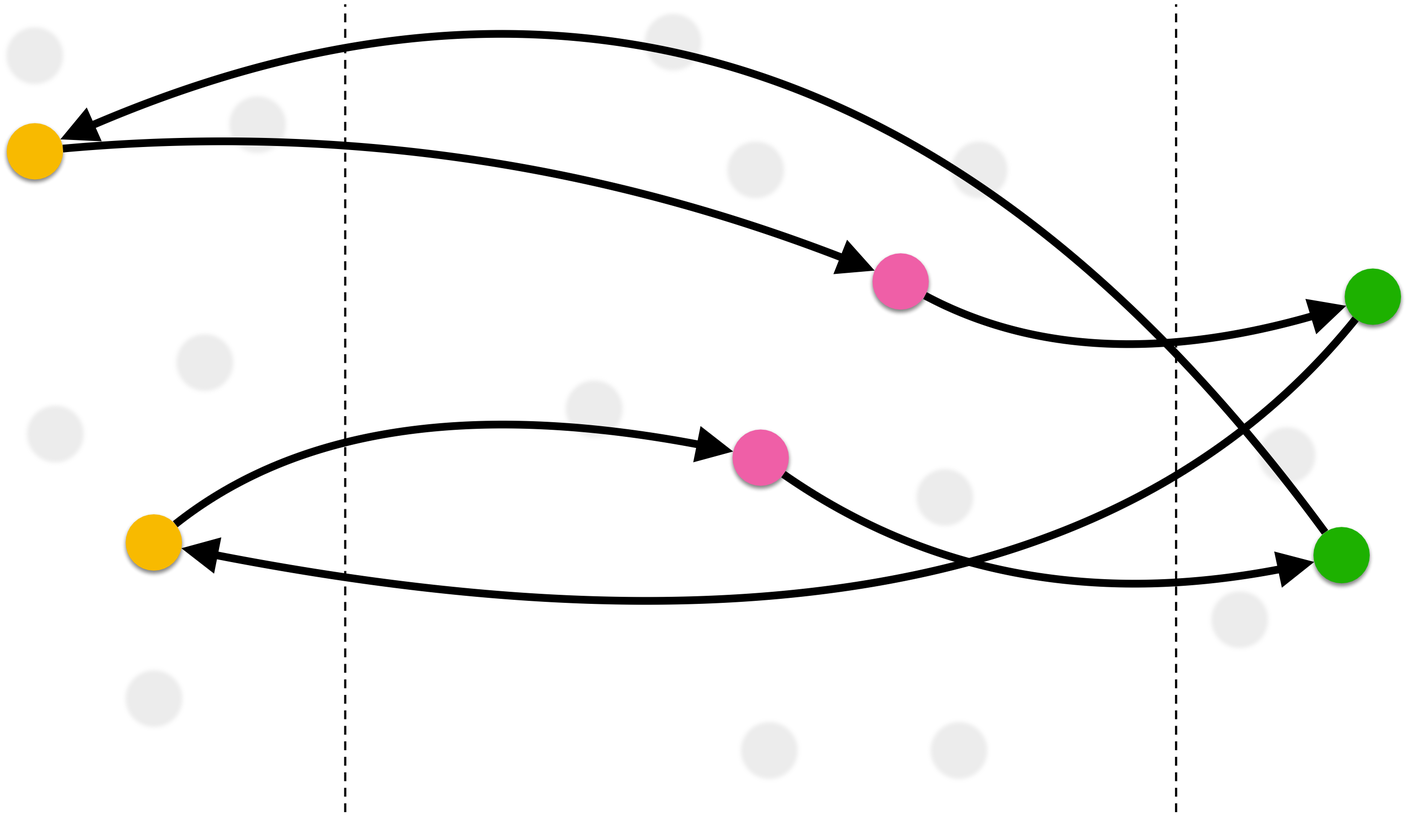
Privacy-Preserving Constraint Solving



Privacy-Preserving Constraint Solving



Privacy-Preserving Constraint Solving

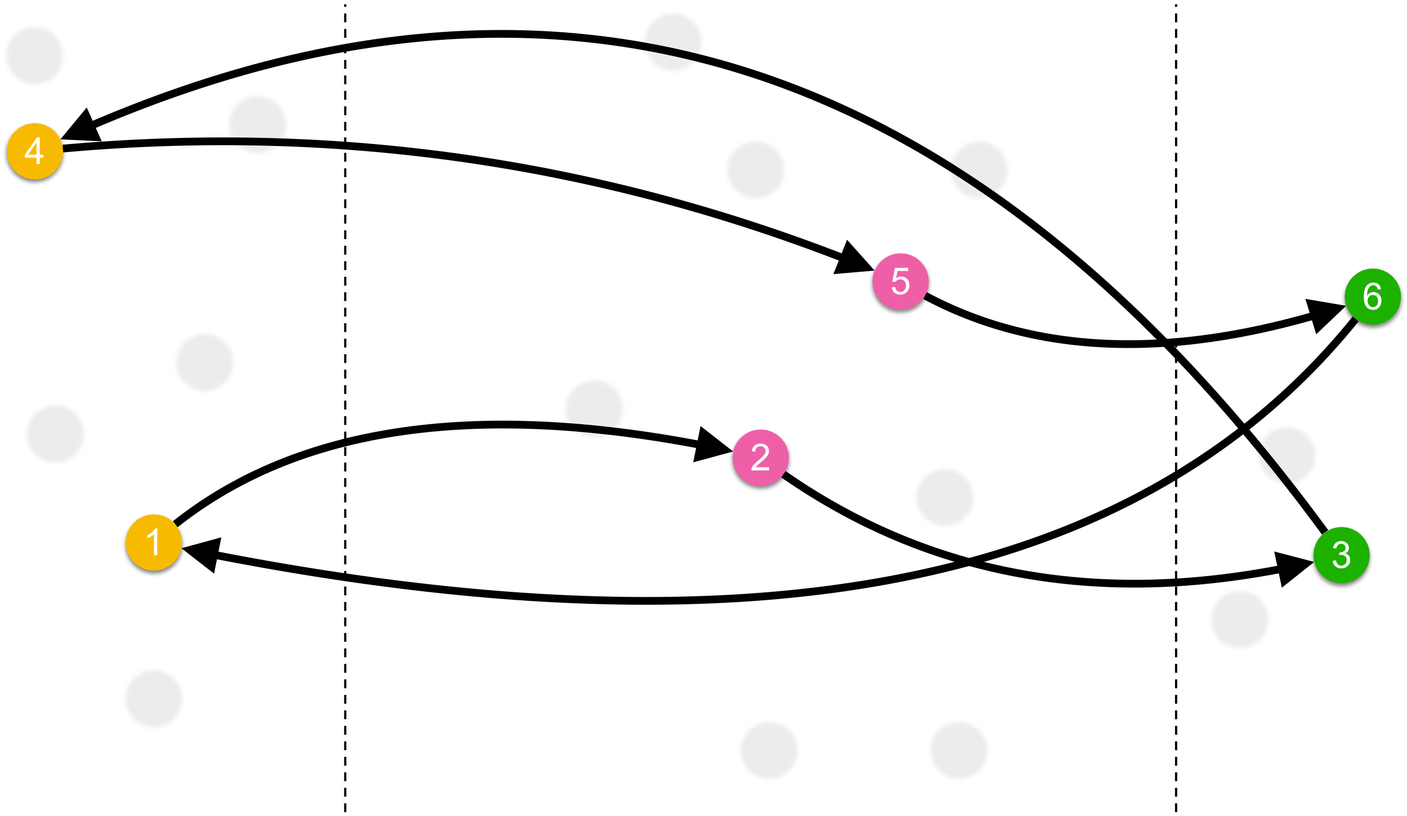


Is there a money laundering ring?

Bank A **Bank B** **Bank C**

Privacy-Preserving Constraint Solving

Is there a money laundering ring?



Bank A **Bank B** **Bank C**

1 2 3 4 5 6

Introduction to the MaxSAT Problem

$$F = \underbrace{(x \vee y)}_{\text{Hard Constraints}} \wedge \underbrace{(x \vee \neg y)}_{\text{Soft Constraints}} \wedge \underbrace{\neg x}_{\text{Soft Constraints}}$$

Goal: Find a model that satisfies all the hard constraints and as many as possible soft constraints

Introduction to the MaxSAT Problem

$$F = \underbrace{(x \vee y)}_{\textbf{HC}} \wedge \underbrace{(x \vee \neg y)}_{\textbf{6}} \wedge \underbrace{\neg x}_{\textbf{3}}$$

Weighted MaxSAT

- Goal: Maximize the combined weight

MaxSAT in Debugging

Buggy Code

```
int Adx[3];
int example(int idx) {
  if (idx != 1) {
    idx = 2;
  } else {
    idx = idx + 2;
  }
  return A[idx];
}
```

$$idx_1 = 1 \vee idx_2 = 2$$

$$idx_1 \neq 1 \vee idx_2 = idx_1 + 2$$

$$idx_2 < 3$$

MaxSAT in Debugging

```
int Adx[3];
int example(int idx) {
  if (idx != 1) {
    idx = 2;
  } else {
    idx = idx + 2;
  }
  return A[idx];
}
```

- Postcondition: $idx_2 < 3$
- Failing test case: $idx_1 = 1$

Encoding the program a formula (SSA)

Program Constraints:

$$\boldsymbol{idx_1 = 1 \vee idx_2 = 2}$$

$$\boldsymbol{idx_1 \neq 1 \vee idx_2 = idx_1 + 2}$$

$$\boldsymbol{idx_2 < 3}$$

Assuming failing test case: $idx_1 = 1$

$$F = (i_1 = 1) \wedge (i_1 = 1 \vee i_2 = 2) \wedge (i_1 \neq 1 \vee i_2 = i_1 + 2) \wedge (i_2 < 3)$$

MaxSAT as a Debugging Tool

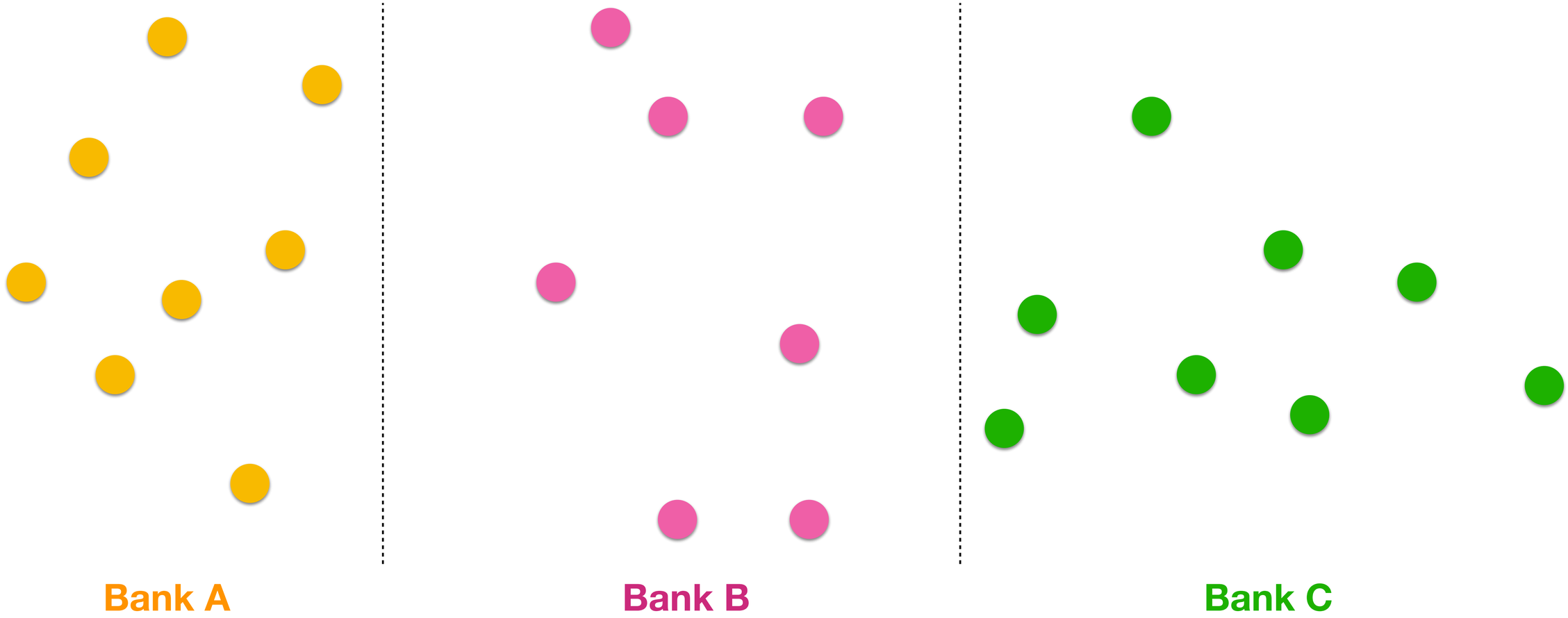
We want:

- HC: Test Input & Final Assertion
- SC: Program Constraints

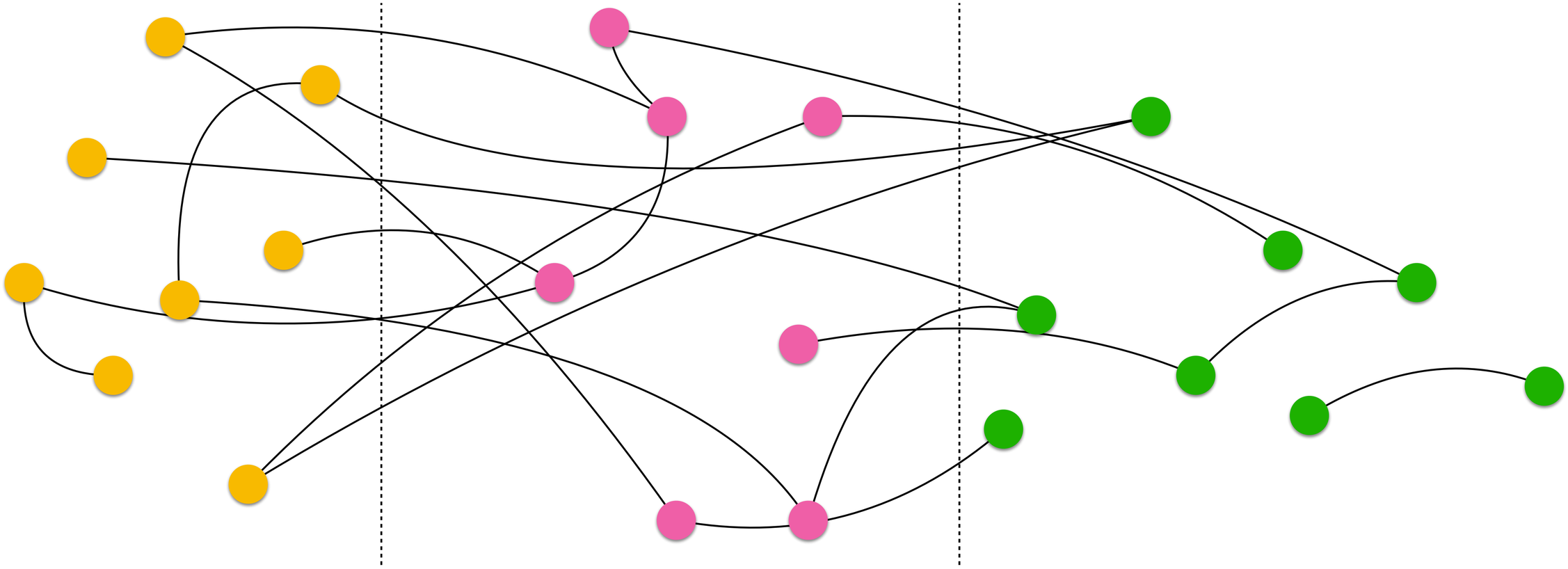
$$F = (i_1 = 1) \wedge (i_1 = 1 \vee i_2 = 2) \wedge (i_1 \neq 1 \vee i_2 = i_1 + 2) \wedge (i_2 < 3)$$

Privacy-Preserving Data Consolidation

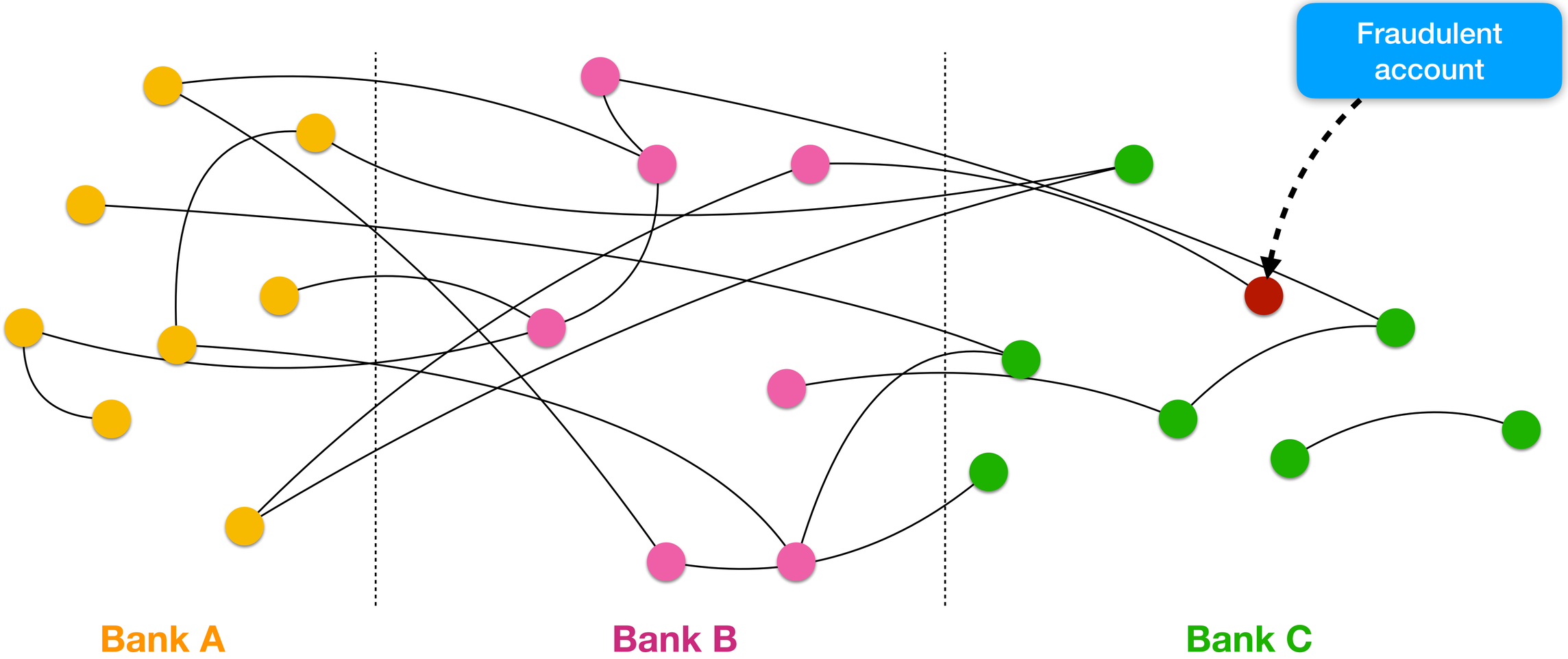
Privacy-Preserving Constraint Solving



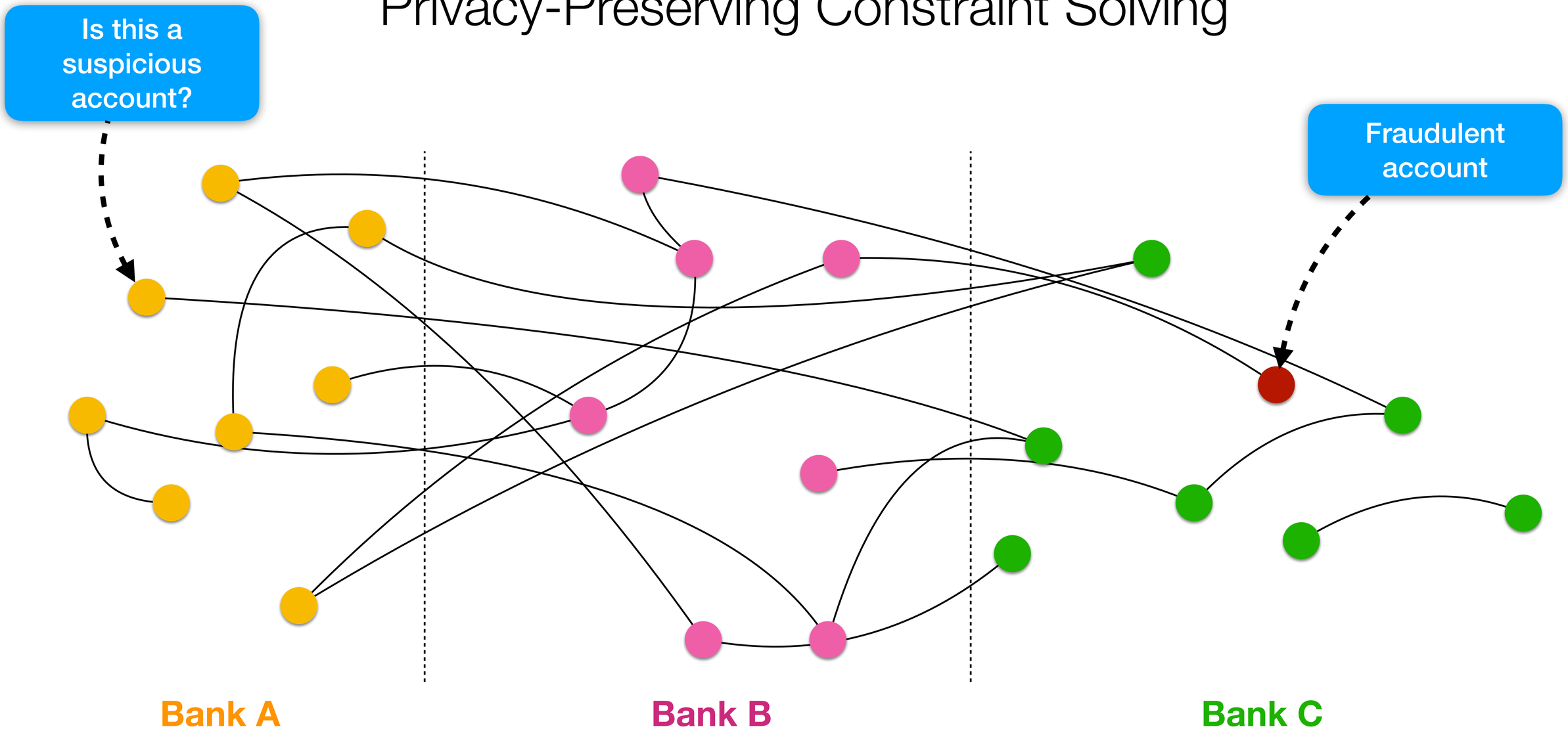
Privacy-Preserving Constraint Solving



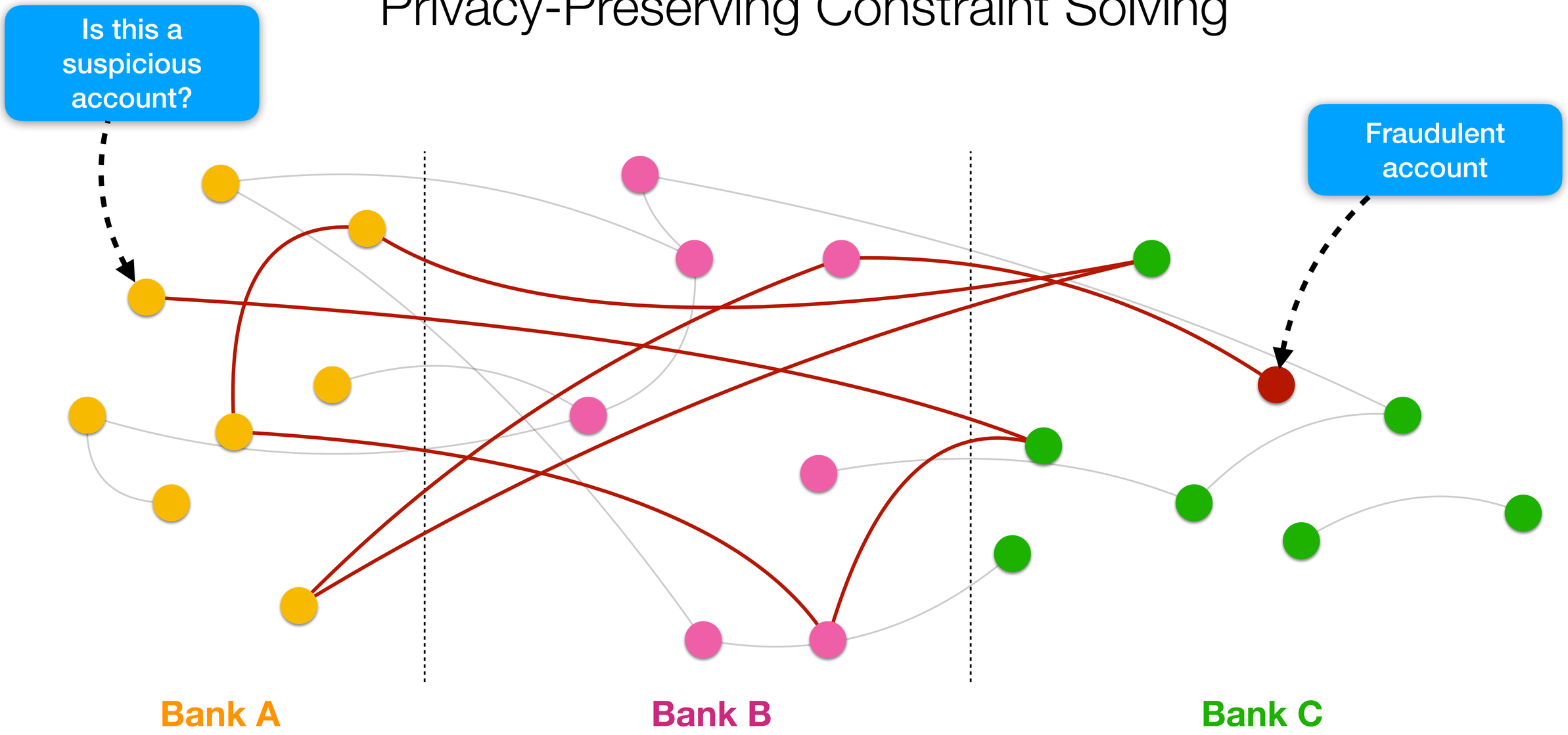
Privacy-Preserving Constraint Solving



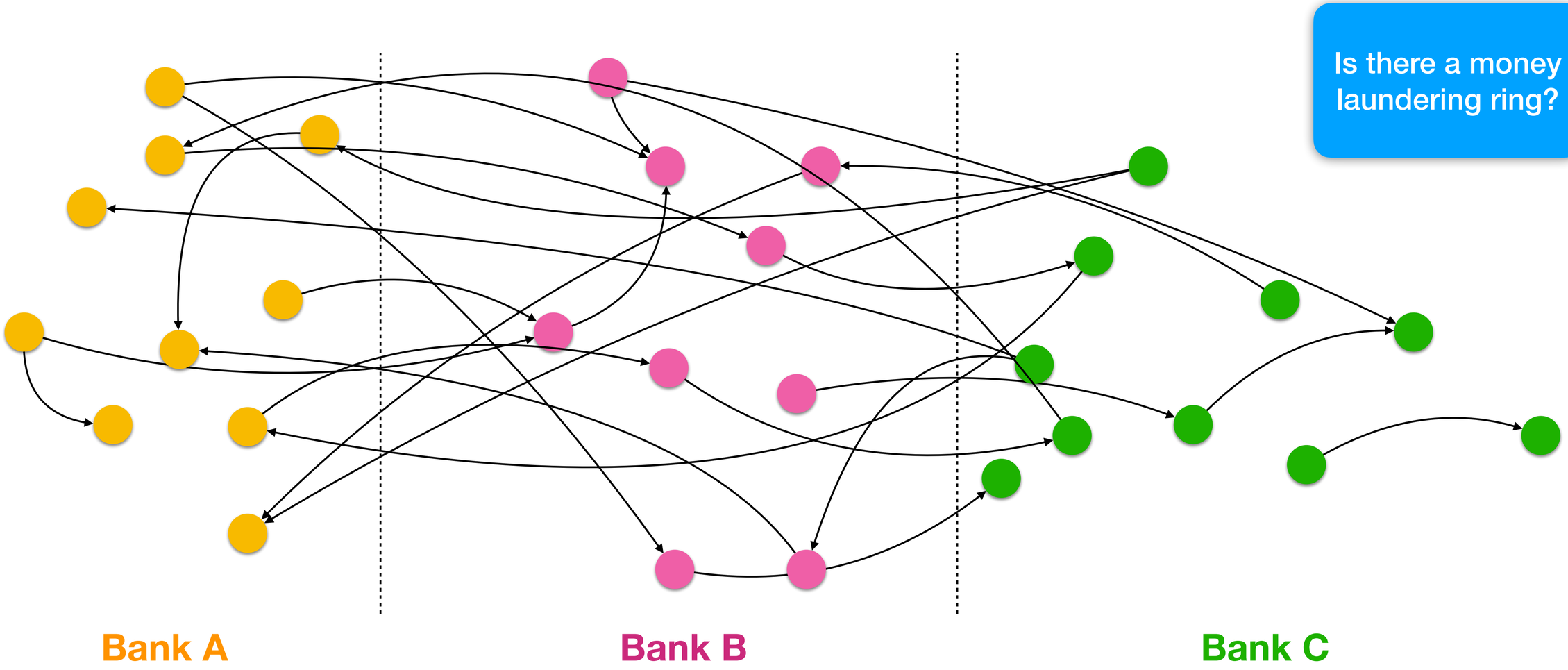
Privacy-Preserving Constraint Solving



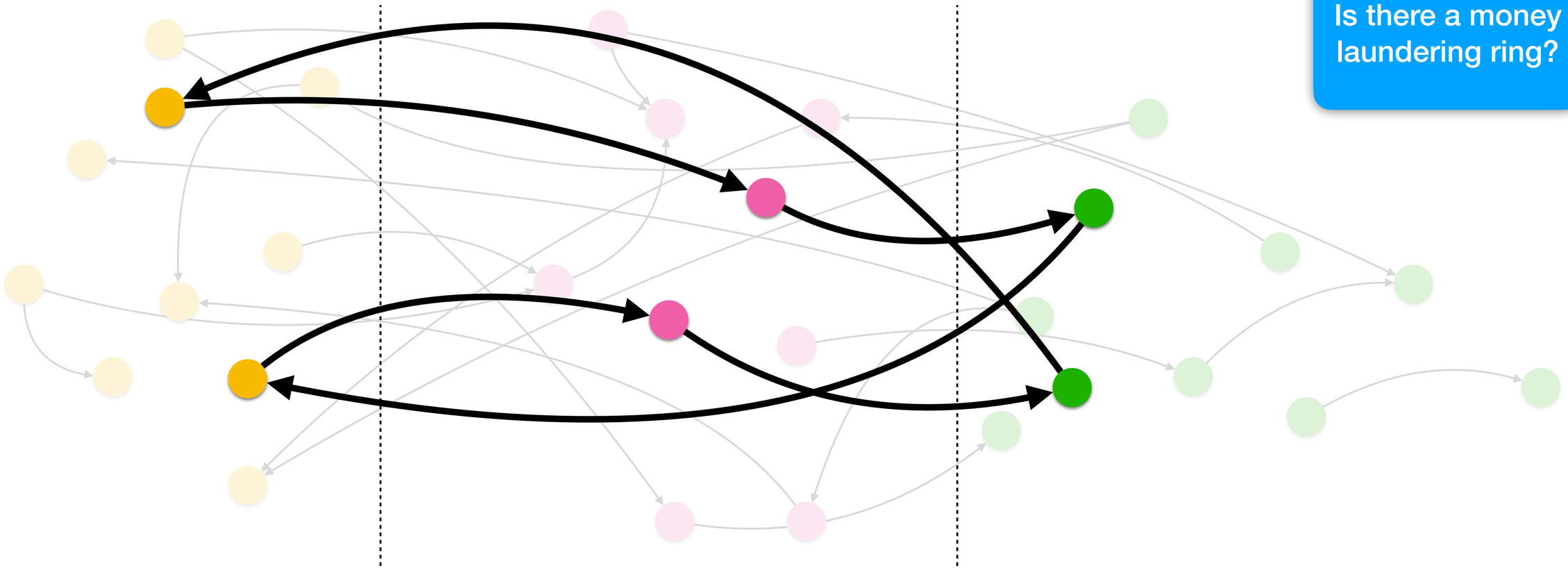
Privacy-Preserving Constraint Solving



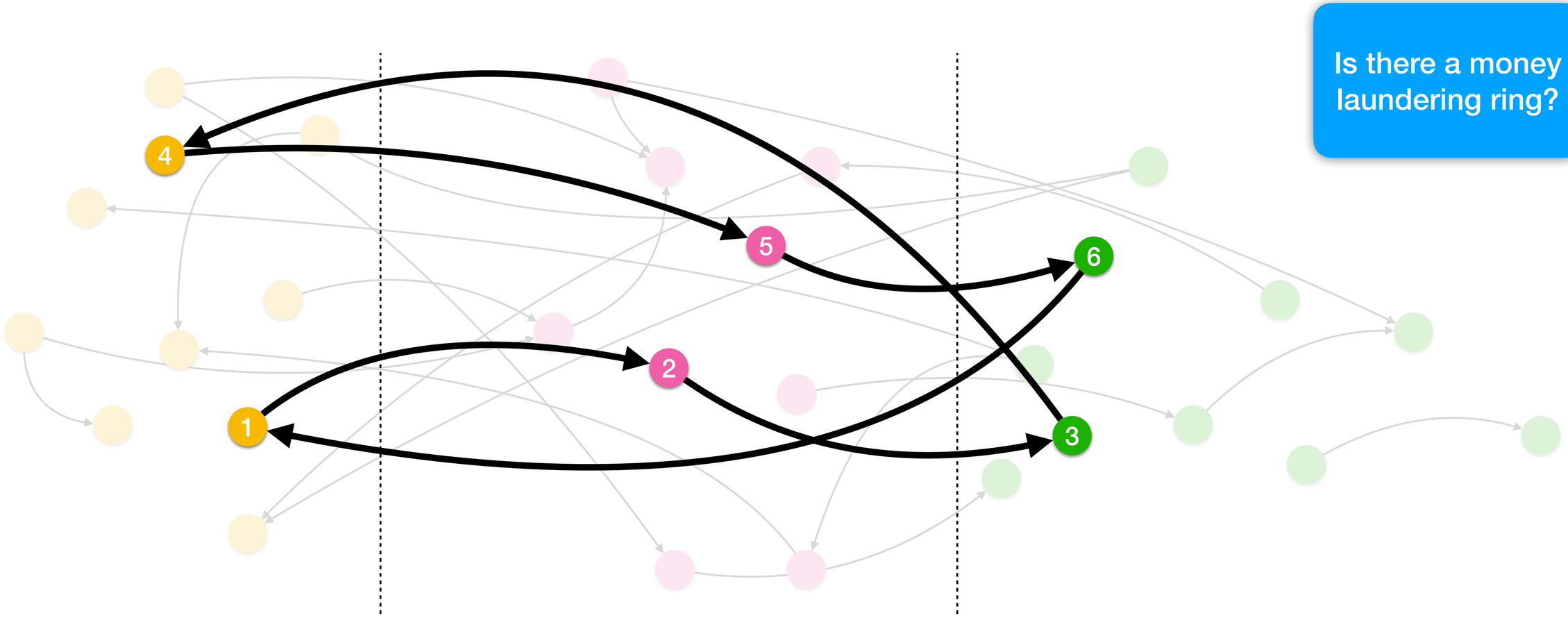
Privacy-Preserving Constraint Solving



Privacy-Preserving Constraint Solving



Privacy-Preserving Constraint Solving



Privacy-Preserving Constraint Solving

Formula will make sure that:

1. every position is filled by exactly one node
2. no node appears twice
3. any two consecutive nodes are adjacent in the graph

Privacy-Preserving Constraint Solving

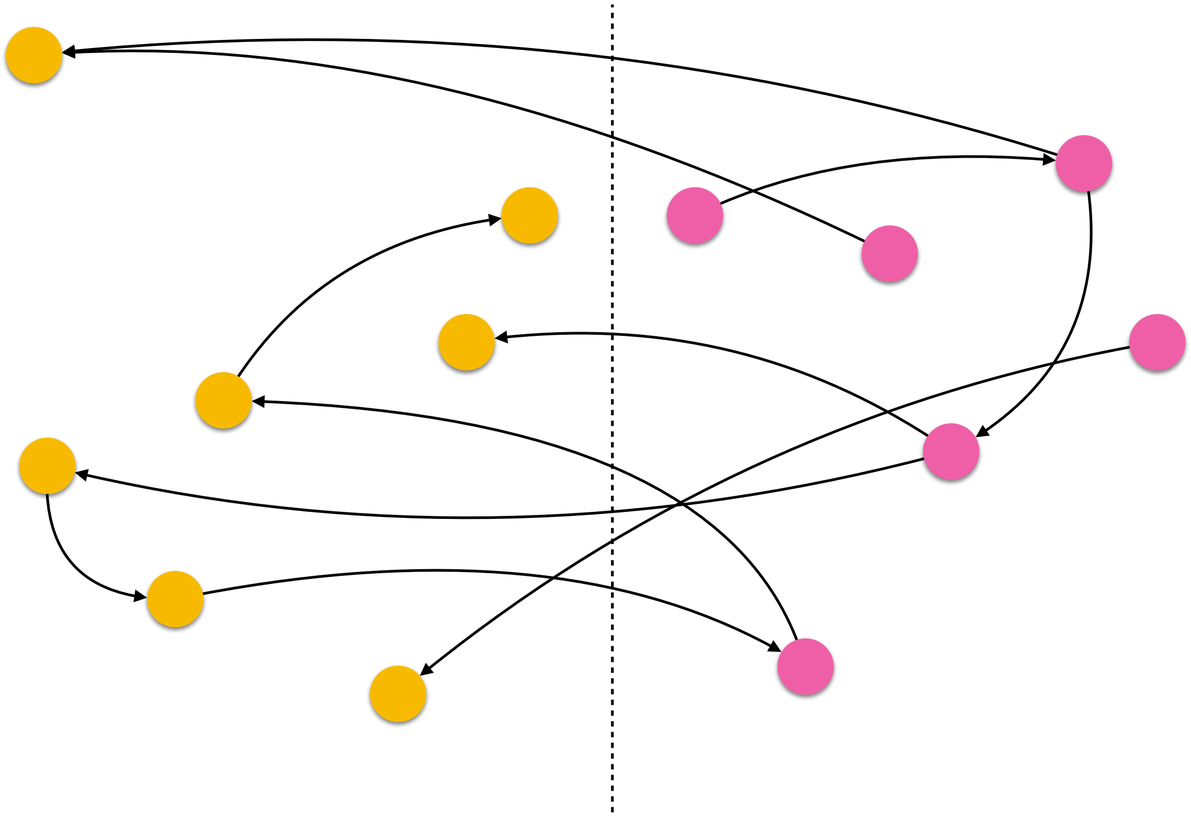
Formula will make sure that:

1. every position is filled by exactly one node
2. no node appears twice
3. any two consecutive nodes are adjacent in the graph

1,000x slowdown in private!!

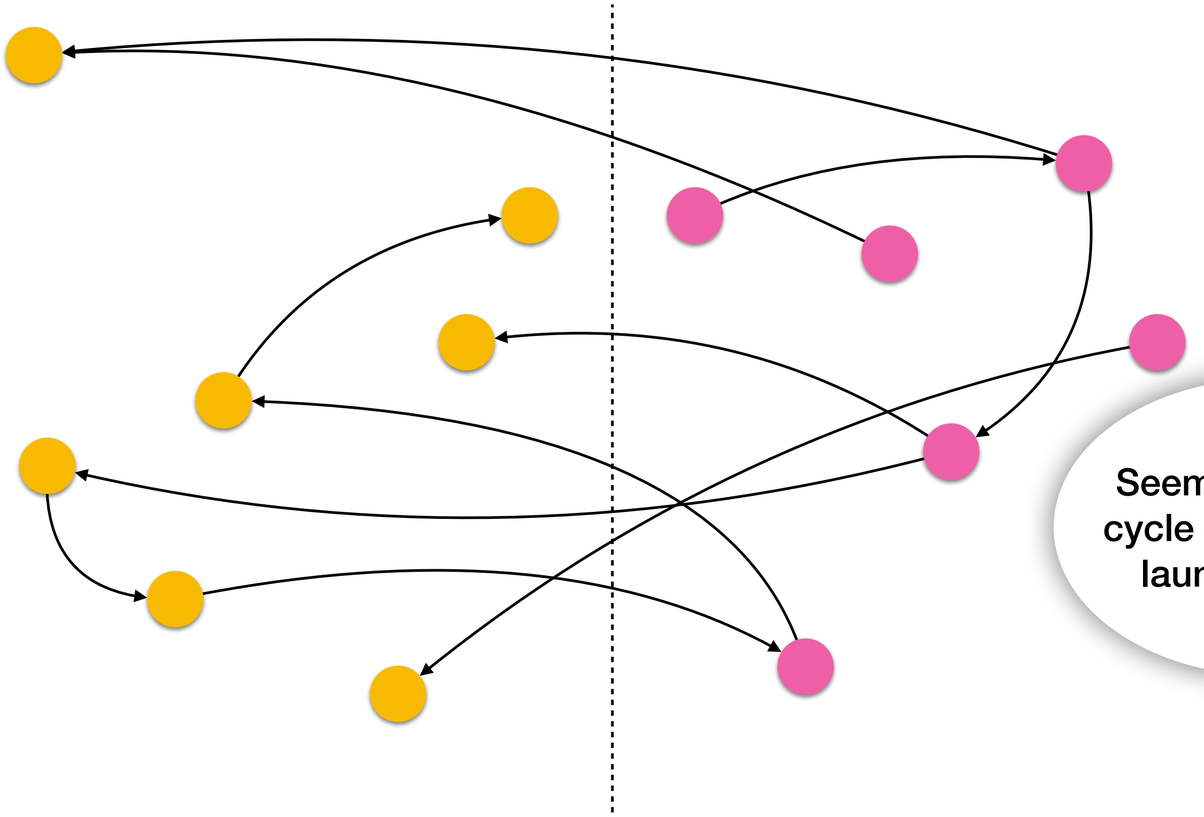
Privacy-Preserving Data Consolidation

Privacy-Preserving Data Consolidation



Bank A **Bank B**

Privacy-Preserving Data Consolidation

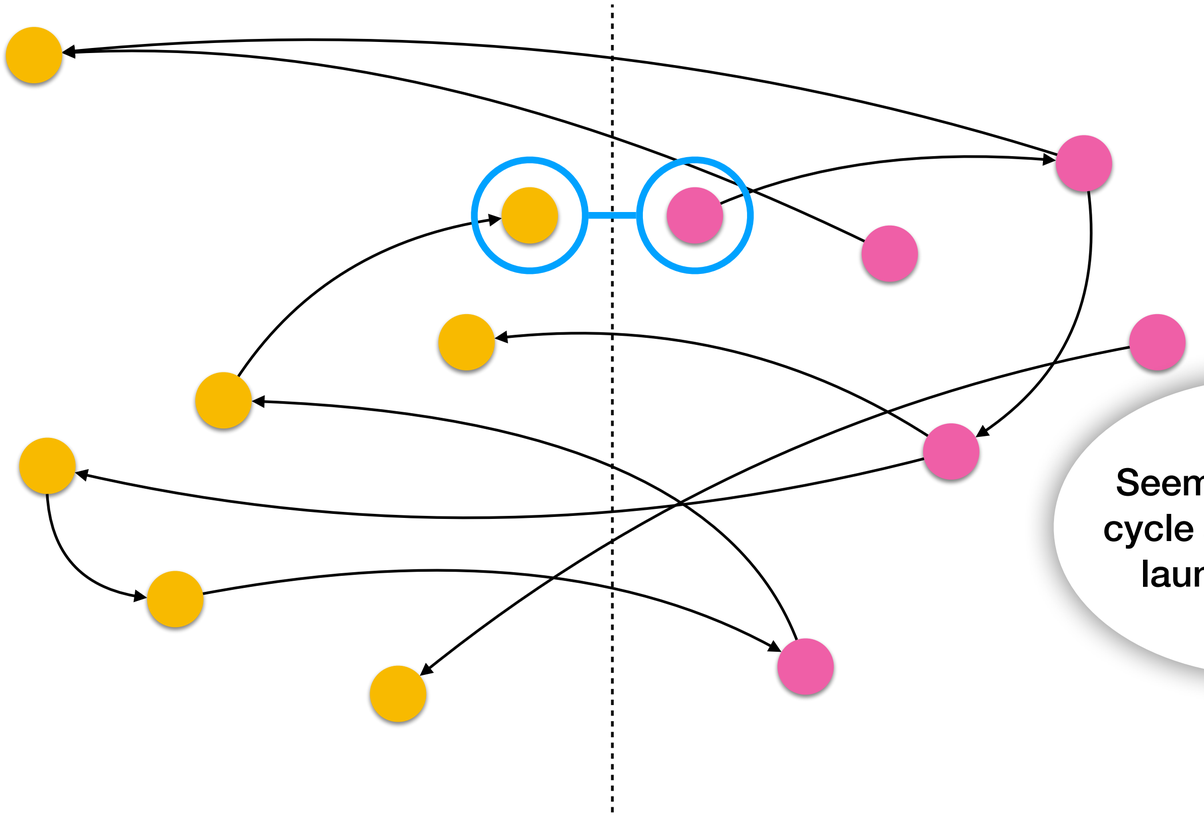


Is there a money laundering ring?

Seems there is no cycle so there is no laundering ring

Bank A **Bank B**

Privacy-Preserving Data Consolidation



Is there a money laundering ring?

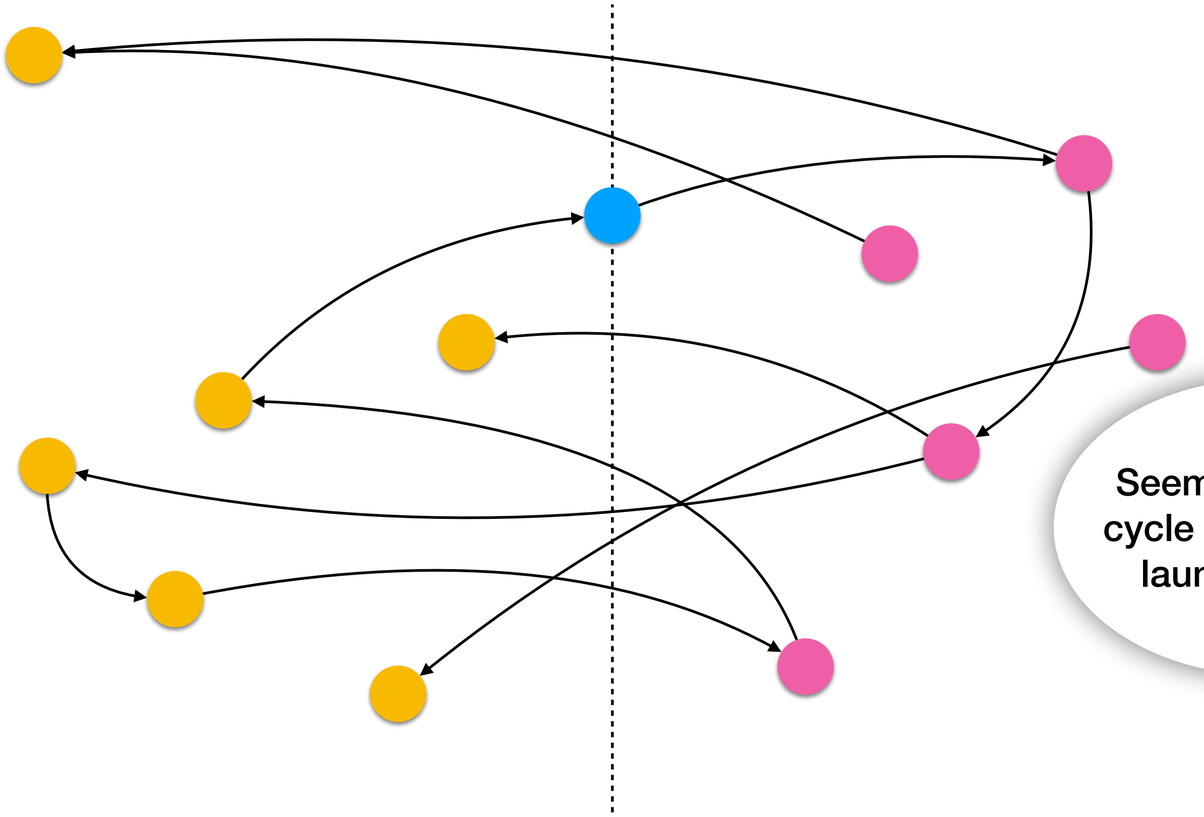
Seems there is no cycle so there is no laundering ring

Bank A

Bank B

I think these two clients might be the same. Is there a ring then?

Privacy-Preserving Data Consolidation



Is there a money laundering ring?

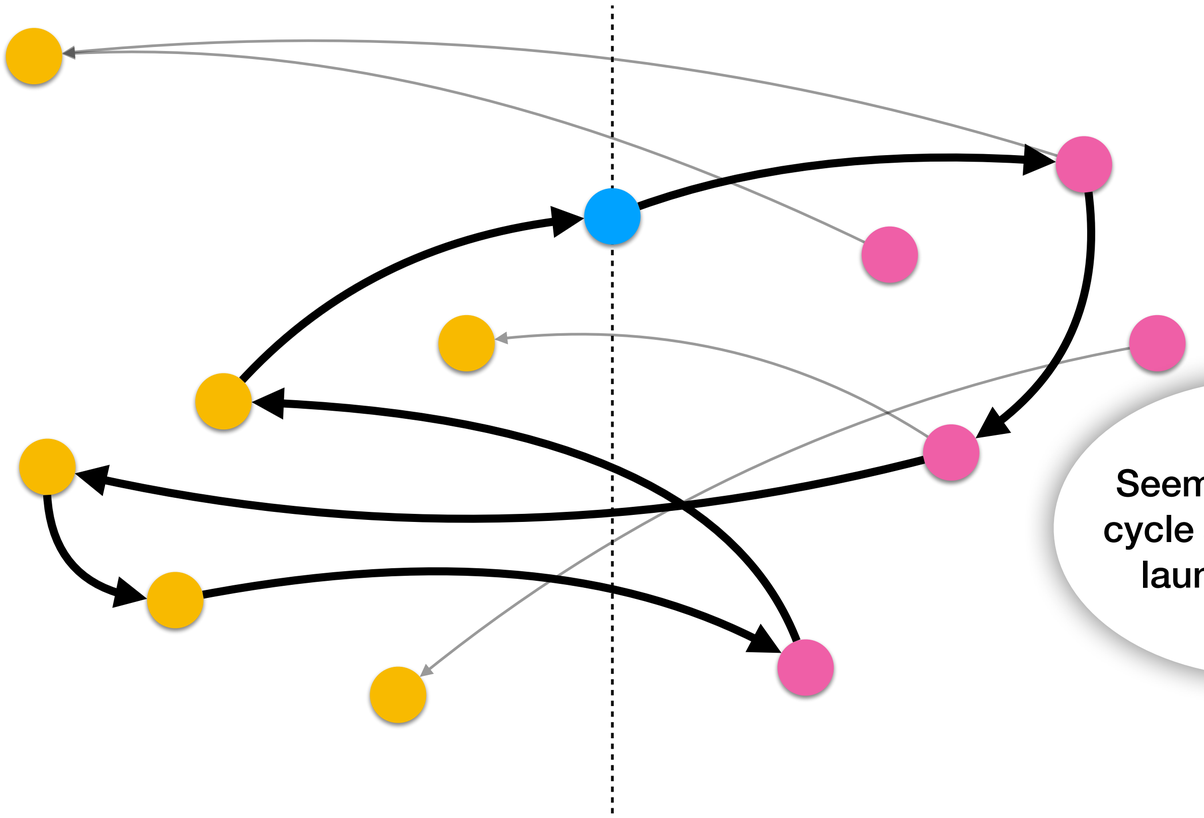
Seems there is no cycle so there is no laundering ring

Bank A

Bank B

I think these two clients might be the same. Is there a ring then?

Privacy-Preserving Data Consolidation



Is there a money laundering ring?

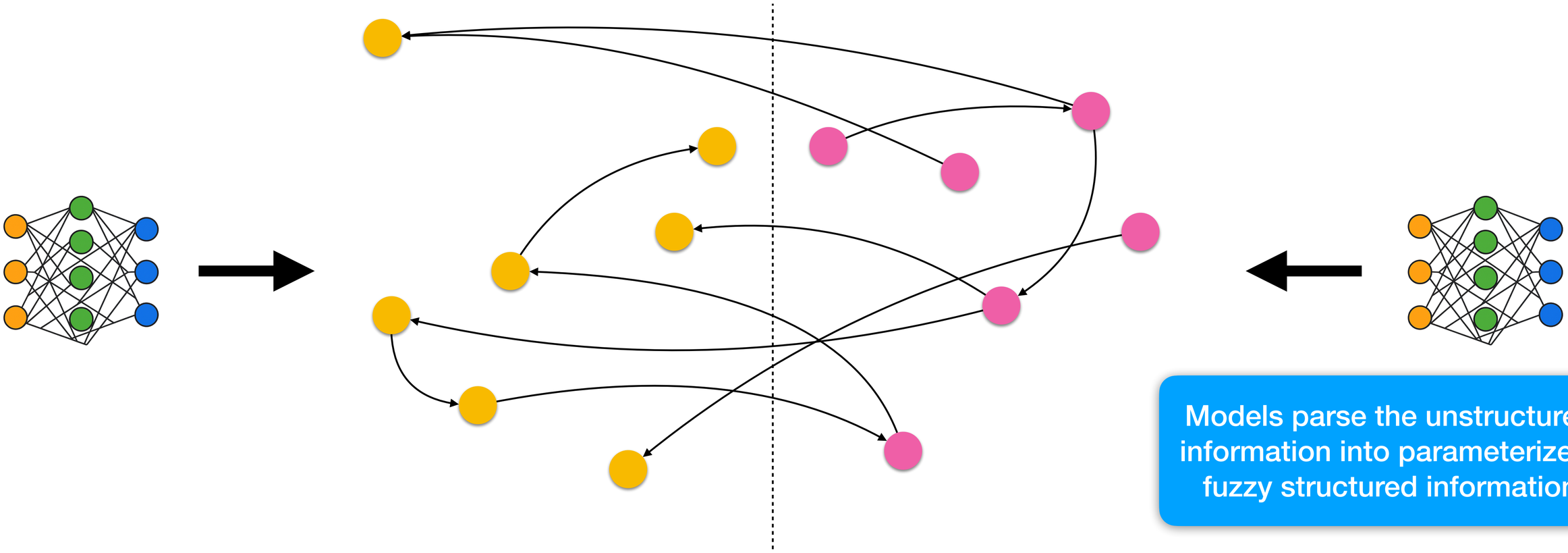
Seems there is no cycle so there is no laundering ring

Bank A

Bank B

I think these two clients might be the same. Is there a ring then?

Privacy-Preserving Data Consolidation

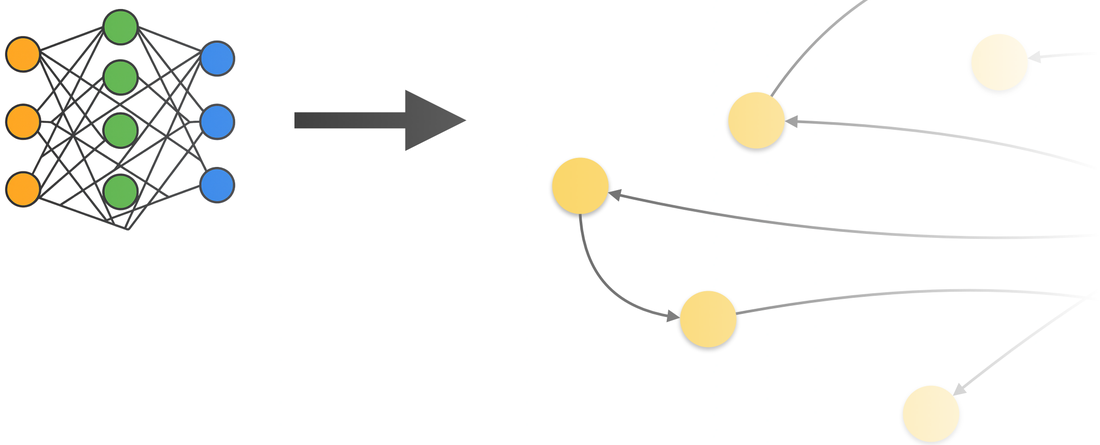


Bank A

Bank B

Models parse the unstructured information into parameterized/ fuzzy structured information

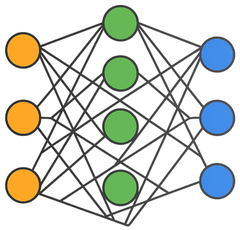
Privacy-Preserving Data Consolidation

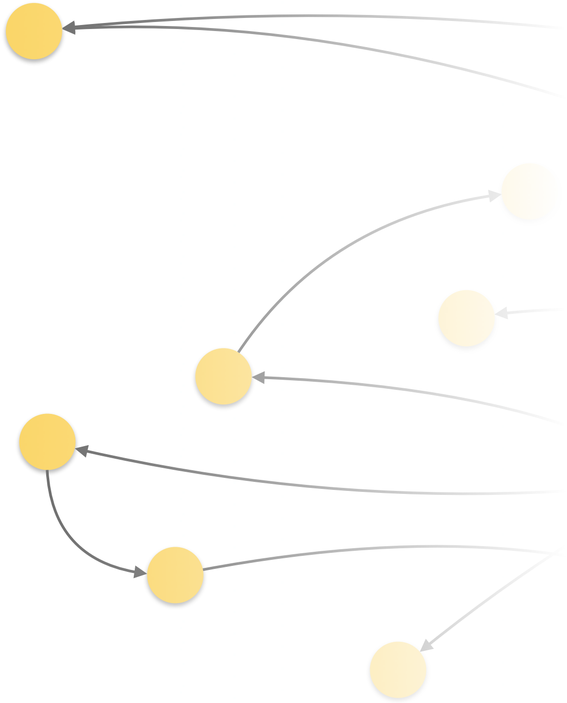


Bank A

On March 12, 2023, at approximately 09:47 CST, **Alicya Quays**, residing at 408 Jefferson Street, **Paris**, Texas 75460, initiated an outbound monetary transaction via First East Bank, branch ID 1427. The transaction was processed through the bank's standard ACH protocol and involved a transfer amount of $18,750.00 USD. The designated recipient account was held at Lone Star Materials, LLC, based in Austin, TX, account number ending in 9241. Verification of sender identity was completed via government-issued ID and signature match. The transaction was authorized and cleared without flag under the bank's AML thresholds and logged under transaction ID #AQRX031223-1427TX.

Privacy-Preserving Data Consolidation





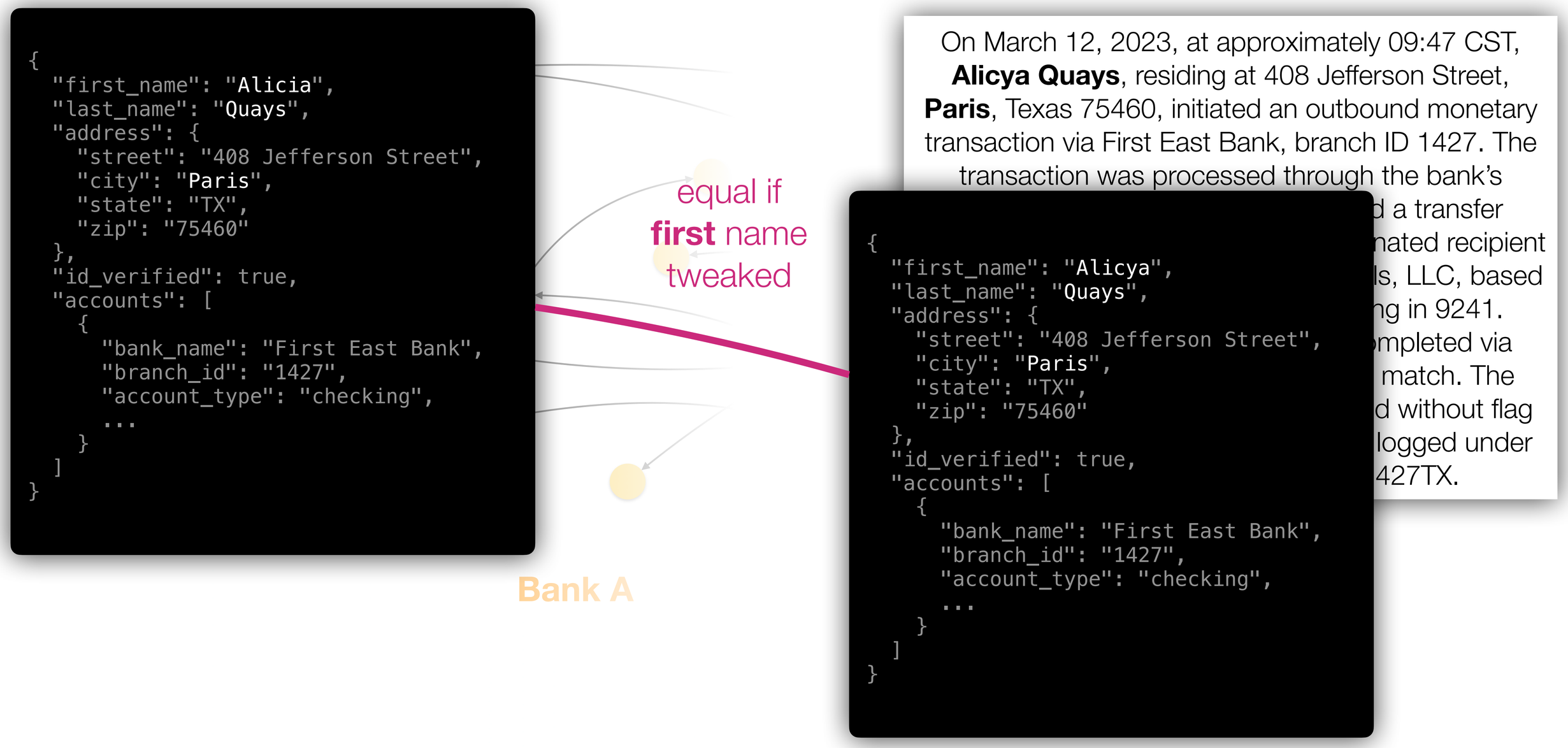
Bank A

On March 12, 2023, at approximately 09:47 CST, **Alicya Quays**, residing at 408 Jefferson Street, **Paris**, Texas 75460, initiated an outbound monetary transaction via First East Bank, branch ID 1427. The transaction was processed through the bank's

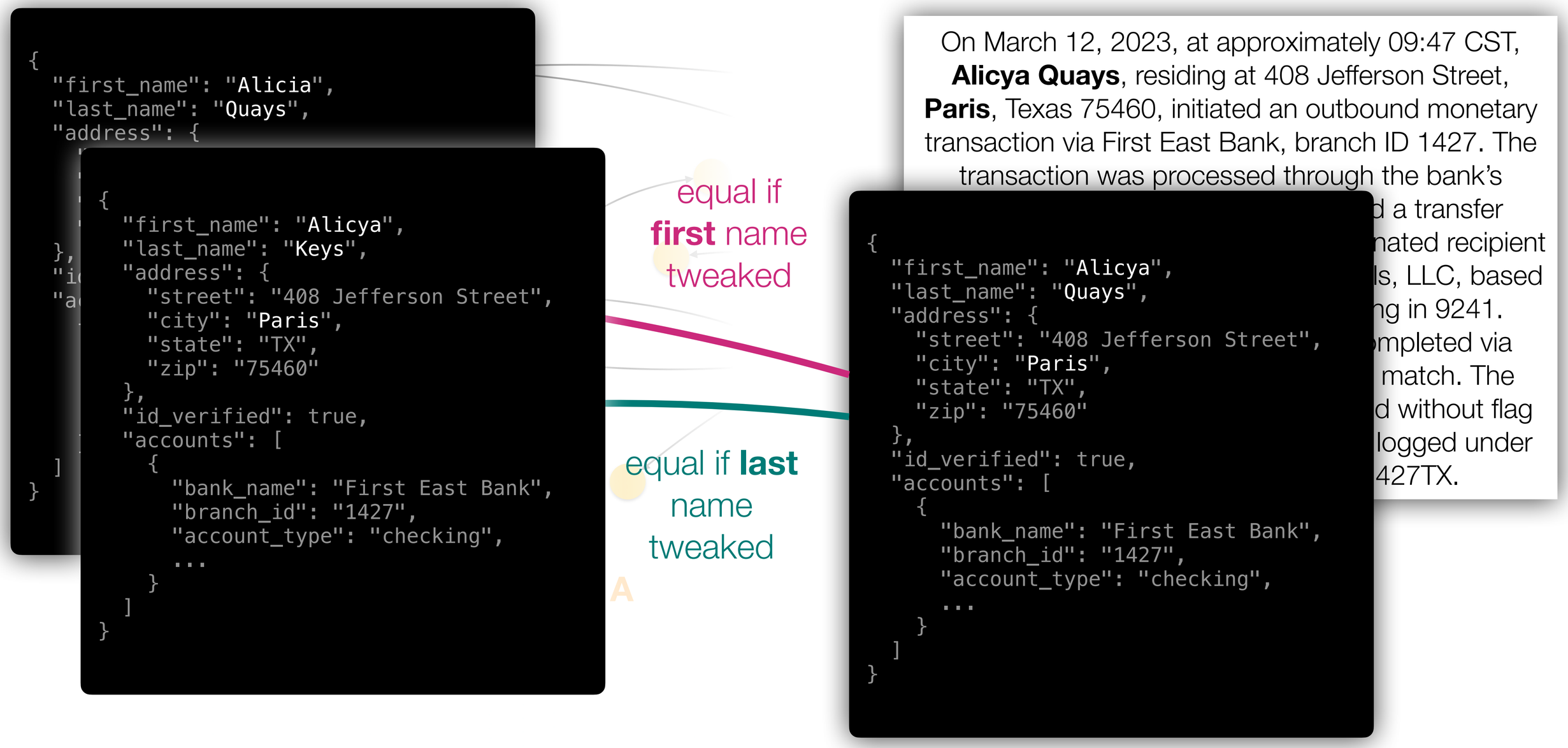
d a transfer
nated recipient
ls, LLC, based
ng in 9241.
mpleted via
match. The
d without flag
logged under
427TX.


```
{
  "first_name": "Alicya",
  "last_name": "Quays",
  "address": {
    "street": "408 Jefferson Street",
    "city": "Paris",
    "state": "TX",
    "zip": "75460"
  },
  "id_verified": true,
  "accounts": [
    {
      "bank_name": "First East Bank",
      "branch_id": "1427",
      "account_type": "checking",
      ...
    }
  ]
}
```

Privacy-Preserving Data Consolidation



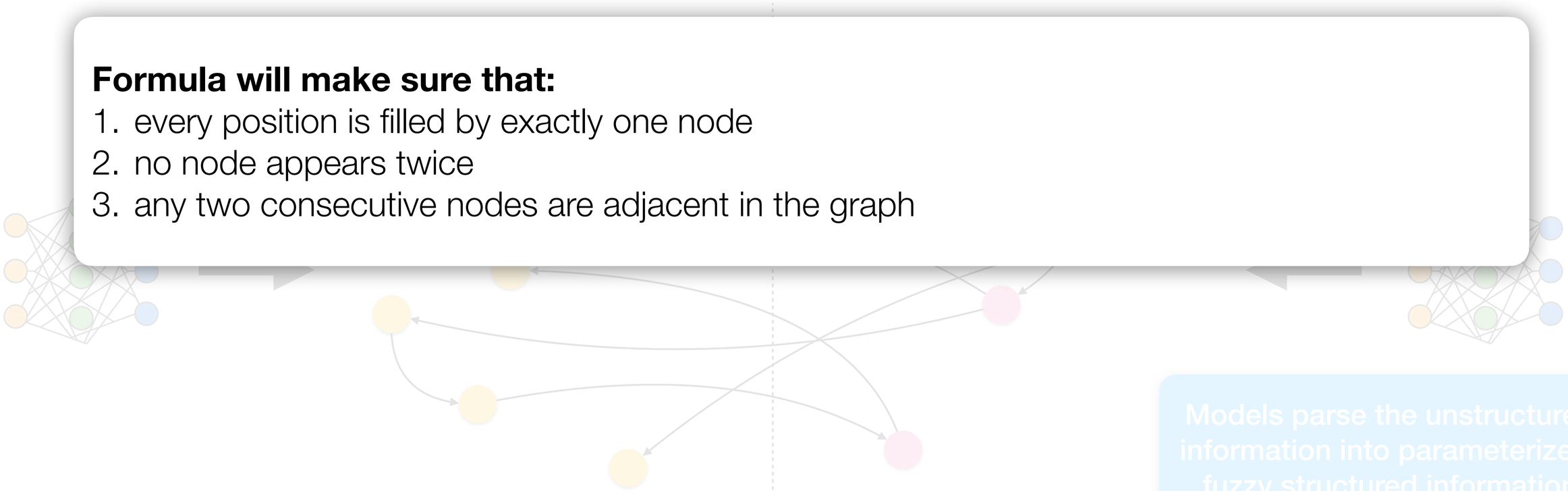
Privacy-Preserving Data Consolidation



Privacy-Preserving Data Consolidation

Formula will make sure that:

1. every position is filled by exactly one node
2. no node appears twice
3. any two consecutive nodes are adjacent in the graph



Models parse the unstructured information into parameterized/ fuzzy structured information

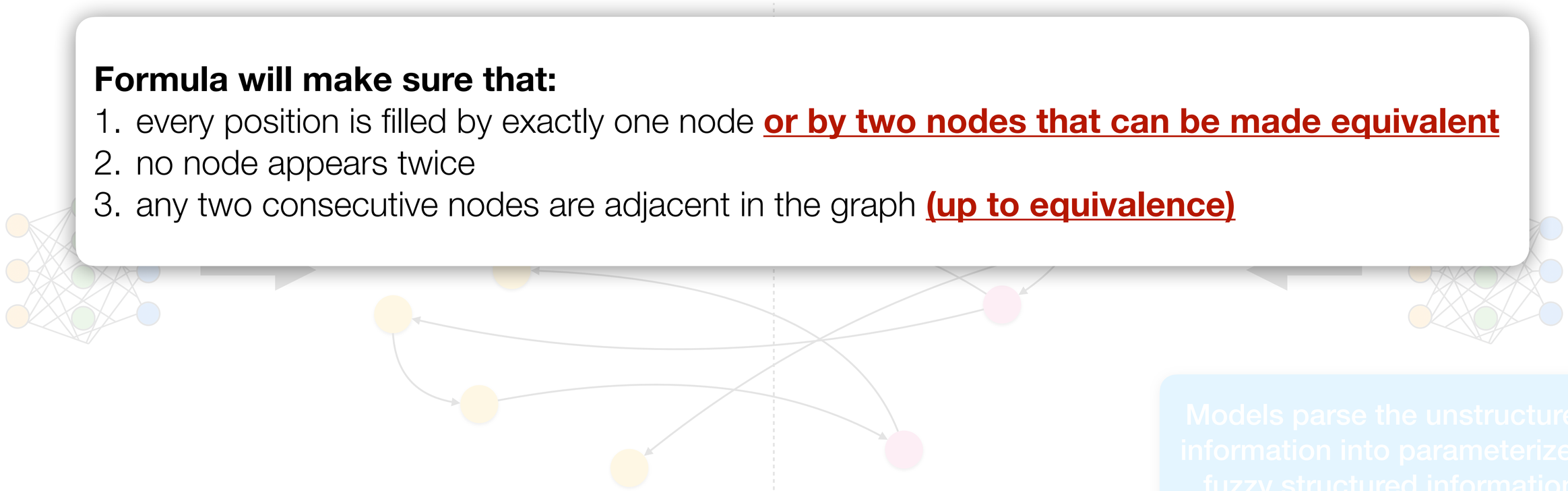
Bank A

Bank B

Privacy-Preserving Data Consolidation

Formula will make sure that:

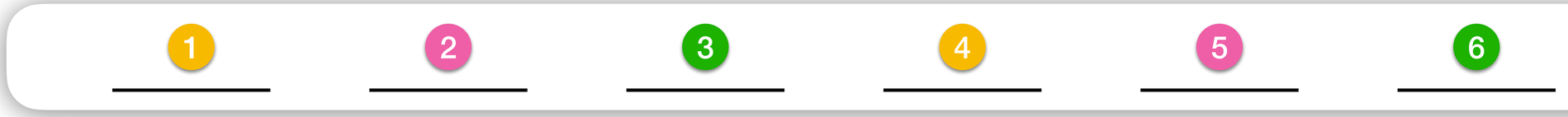
1. every position is filled by exactly one node **or by two nodes that can be made equivalent**
2. no node appears twice
3. any two consecutive nodes are adjacent in the graph **(up to equivalence)**



Models parse the unstructured information into parameterized/ fuzzy structured information

Bank A

Bank B



Privacy-Preserving Data Consolidation

Formula will make sure that:

1. every position is filled by exactly one node **or by two nodes that can be made equivalent**
2. no node appears twice
3. any two consecutive nodes are adjacent in the graph **(up to equivalence)**

encoded as a soft constraint that can be "**broken**" at a cost

nstructured information into parameterized/ fuzzy structured information

Bank A

Bank B

Privacy-Preserving Data Consolidation

Formula will make sure that:

1. every position is filled by exactly one node **or by two nodes that can be made equivalent**
2. no node appears twice
3. any two consecutive nodes are adjacent in the graph **(up to equivalence)**

encoded as a soft constraint that can be "**broken**" at a cost

Privacy-Preserving MaxSAT

nstructured information into parameterized/ fuzzy structured information

Bank A

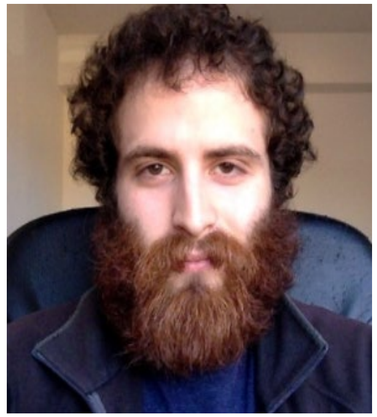
References

- ***Coinductive Proofs of Regular Expression Equivalence in Zero Knowledge***
  John C. Kolesar, Shan Ali, Timos Antonopoulos, Ruzica Piskac, Proc. ACM Program. Lang. 9(OOPSLA2): 357-385 (2025)
- ***ZKSMT: A VM for Proving SMT Theorems in Zero Knowledge***
  Daniel Luick, John Kolesar, Timos Antonopoulos, William R. Harris, James Parker, Ruzica Piskac, Eran Tromer, Xiao Wang, Ning Luo.
  USENIX Security 2024
- ***Privacy-Preserving Regular Expression Matching using Nondeterministic Finite Automata***
  Ning Luo*, Chenkai Weng*, Jaspal Singh, Gefei Tan, Ruzica Piskac, Mariana Raykova.
  ESORICS 2024
- ***Ou: Automating the Parallelization of Zero-Knowledge Protocols***
  Yuyang Sang*, Ning Luo*, Samuel Judson, Ben Chaimberg, Timos Antonopoulos, Ruzica Piskac, Xiao Wang, Zhong Shao
  CCS 2023
- ***Proving UNSAT in Zero Knowledge***
  Ning Luo, Timos Antonopoulos, William Harris, Ruzica Piskac, Eran Tromer, Xiao Wang
  CCS 2022 (**Distinguished paper award**)
- ***ppSAT: Towards Two-Party Private SAT Solving***
  Ning Luo, Samuel Judson, Timos Antonopoulos, Ruzica Piskac, Xiao Wang
  USENIX Security 2022
- ***Privacy Preserving CTL Model Checking through Oblivious Graph Algorithms***
  Samuel Judson, Ning Luo, Timos Antonopoulos, Ruzica Piskac
  WPES 2020

Acknowledgements: Amazing Collaborators!

Timos Antonopoulos

Bill Harris



Sam Judson

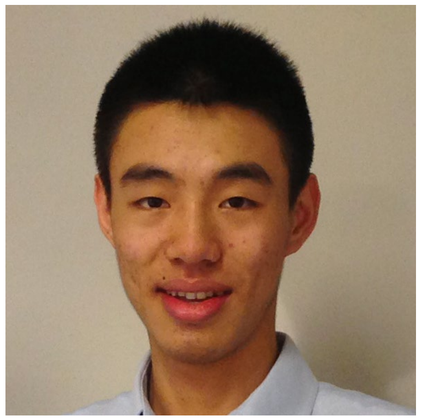
John Kolesar

Daniel Luick

James Parker

Yuyang Sang

Eran Tromer



Xiao Wang

Conclusions

- Enables secure reasoning over sensitive data without revealing private inputs
- Main challenges include scalability, communication overhead, and efficient proof generation
- Combines automated reasoning techniques with cryptography and MPC protocols
- Supports applications in finance, security, compliance checking, …

- **A very exciting novel application of automated reasoning**