Scaling Secure Computation

David Evans
University of Virginia

Oregon Security Day
5 April 2013

http://www.cs.virginia.edu/evans
http://MightBeEvil.com

(De)Motivating Application: “Genetic Dating”

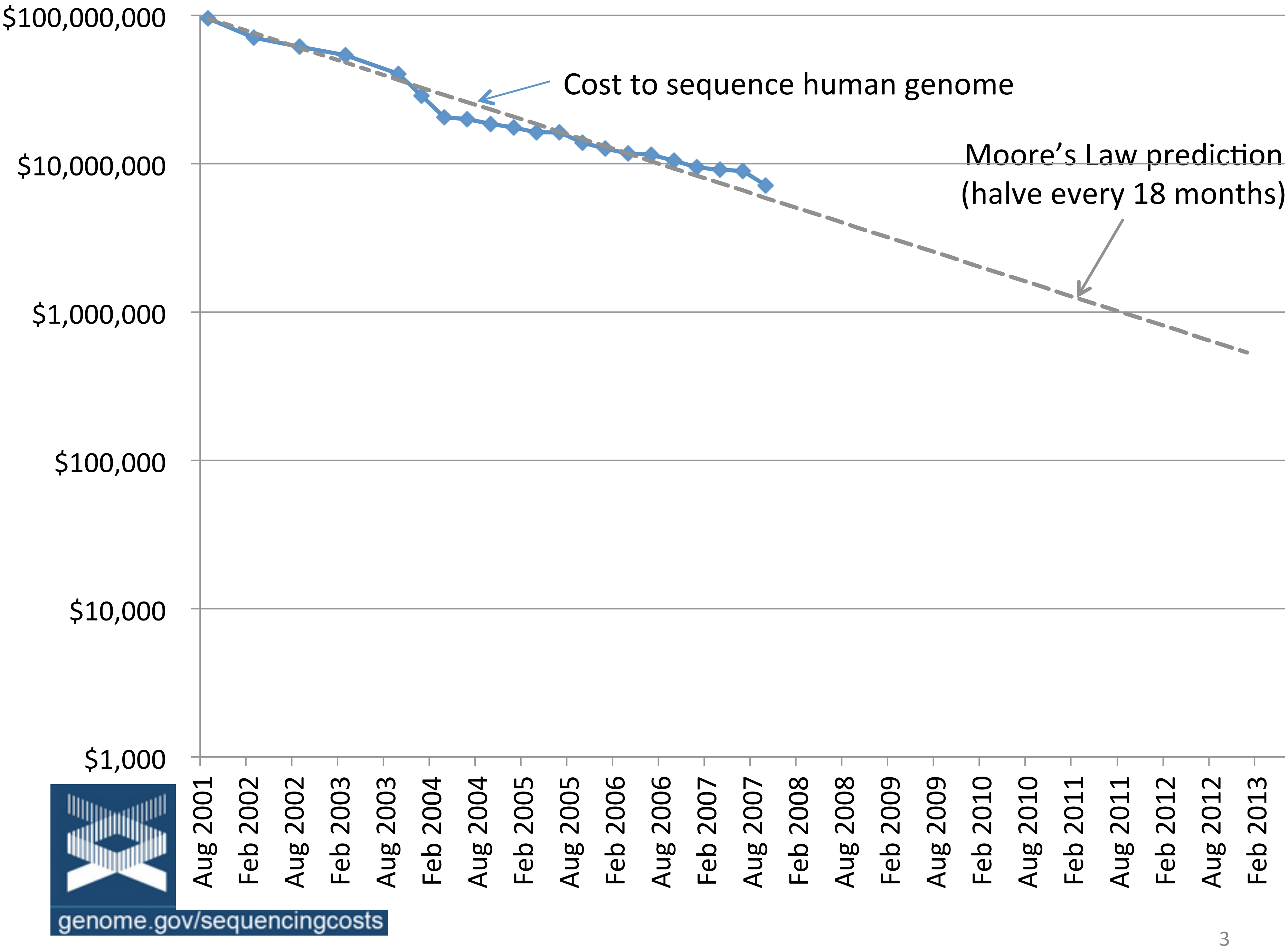
Bob

Alice

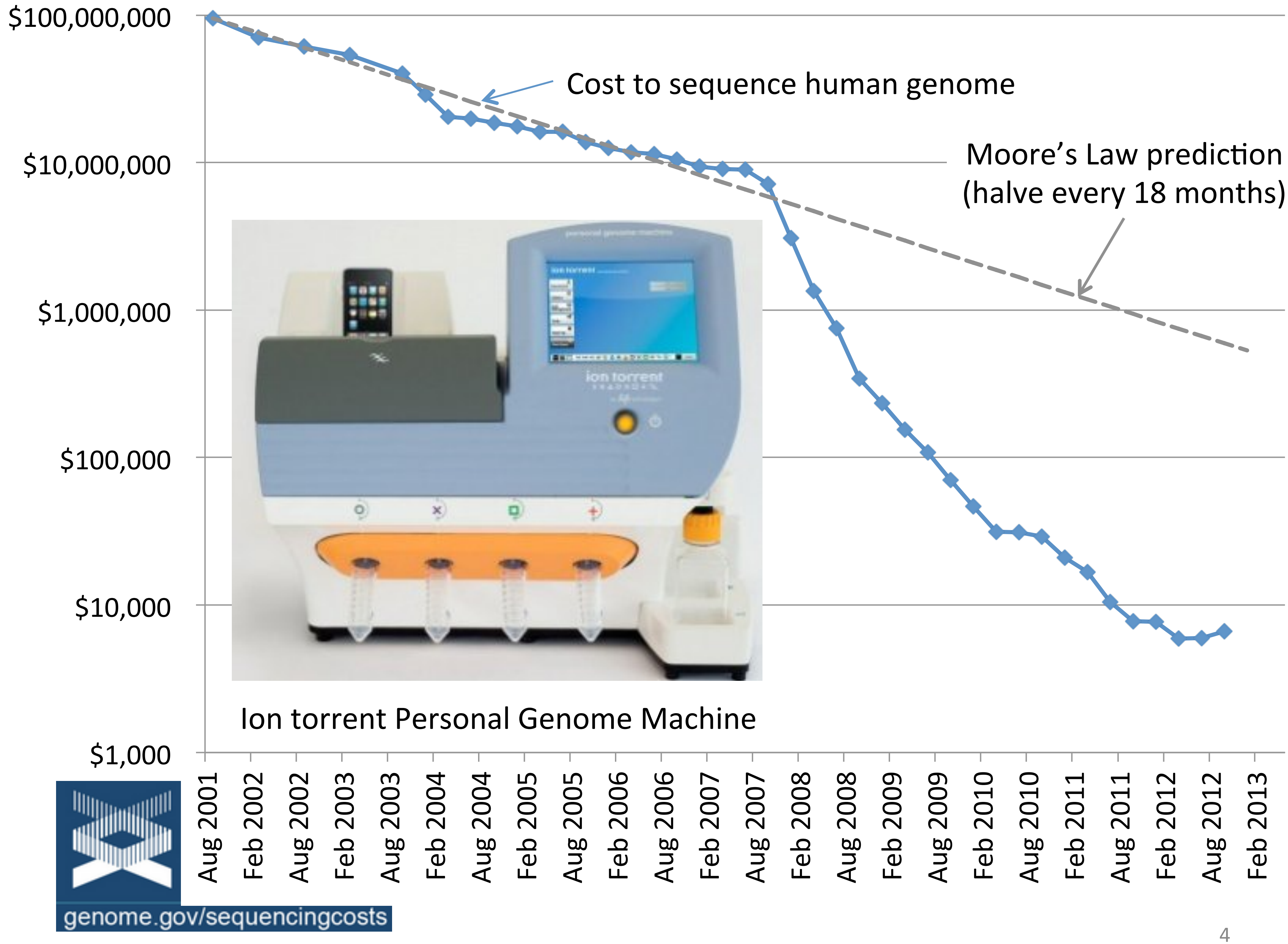
Genome Compatibility Protocol

WARNING!
Don’t Reproduce

WARNING!
Don’t Reproduce



$100,000,000
$10,000,000
$1,000,000
$100,000
$10,000
$1,000
Cost to sequence human genome
Moore's Law prediction
(halve every 18 months)
Aug 2001
Feb 2002
Aug 2002
Feb 2003
Aug 2003
Feb 2004
Aug 2004
Feb 2005
Aug 2005
Feb 2006
Aug 2006
Feb 2007
Aug 2007
Feb 2008
Aug 2008
Feb 2009
Aug 2009
Feb 2010
Aug 2010
Feb 2011
Aug 2011
Feb 2012
Aug 2012
Feb 2013
genome.gov/sequencingcosts


$100,000,000
$10,000,000
$1,000,000
$100,000
$10,000
$1,000
Cost to sequence human genome
Moore's Law prediction
(halve every 18 months)
Ion torrent Personal Genome Machine
Aug 2001
Feb 2002
Aug 2002
Feb 2003
Aug 2003
Feb 2004
Aug 2004
Feb 2005
Aug 2005
Feb 2006
Aug 2006
Feb 2007
Aug 2007
Feb 2008
Aug 2008
Feb 2009
Aug 2009
Feb 2010
Aug 2010
Feb 2011
Aug 2011
Feb 2012
Aug 2012
Feb 2013
genome.gov/sequencingcosts

Year	reference	Technology	Sample	Average Reported Coverage depth (fold)	Reported sequencing consumables cost	Estimated cost per 40-fold coverage
[redacted]	*S4*	Sanger (ABI)	JCV	7	$10,000,000	$57,000,000
[redacted]	*S5*	Roche(454)	JDW	7	$1,000,000	$5,700,000
[redacted]	*S6*	Illumina	NA18507	30	$250,000	$330,000
[redacted]	*S7*	Helicos	SRQ	28	$48,000	$69,000
2009	this work	this work	NA07022	87	$8,005	$3,700
2009	this work	this work	NA19240	63	$3,451	$2,200
2009	this work	this work	NA20431	45	$1,726	$1,500

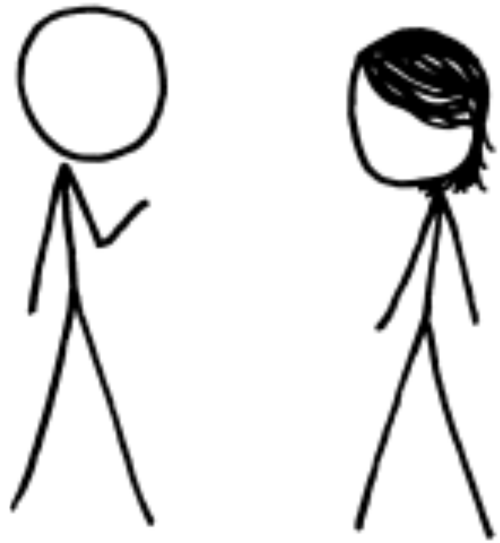
***Human Genome Sequencing Using Unchained Base Reads on Self-Assembling DNA Nanoarrays*.** Radoje Drmanac, Andrew B. Sparks, Matthew J. Callow, Aaron L. Halpern, Norman L. Burns, Bahram G. Kermani, Paolo Carnevali, Igor Nazarenko, Geoffrey B. Nilsen, George Yeung, Fredrik Dahl, Andres Fernandez, Bryan Staker, Krishna P. Pant, Jonathan Baccash, Adam P. Borcherding, Anushka Brownley, Ryan Cedeno, Linsu Chen, Dan Chernikoff, Alex Cheung, Razvan Chirita, Benjamin Curson, Jessica C. Ebert, Coleen R. Hacker, Robert Hartlage, Brian Hauser, Steve Huang, Yuan Jiang, Vitali Karpinchyk, Mark Koenig, Calvin Kong, Tom Landers, Catherine Le, Jia Liu, Celeste E. McBride, Matt Morenzoni, Robert E. Morey, Karl Mutch, Helena Perazich, Kimberly Perry, Brock A. Peters, Joe Peterson, Charit L. Pethiyagoda, Kaliprasad Pothuraju, Claudia Richter, Abraham M. Rosenbaum, Shaunak Roy, Jay Shafto, Uladzislau Sharanhovich, Karen W. Shannon, Conrad G. Sheppy, Michel Sun, Joseph V. Thakuria, Anne Tran, Dylan Vu, Alexander Wait Zaranek, Xiaodi Wu, Snezana Drmanac, Arnold R. Oliphant, William C. Banyai, Bruce Martin, Dennis G. Ballinger, George M. Church, Clifford A. Reid. ***Science*, January 2010.**

Dystopia

Personalized Medicine

Secure Two-Party Computation

Bob's Genome: ACTG…
Markers (~1000): [0,1, …, 0]



Alice's Genome: ACTG…
Markers (~1000): [0, 0, …, 1]

Bob

Alice

$$x = f(g_A, g_B)$$

Can Alice and Bob compute a function of their private data, without exposing anything about their data besides the result?

Secure Function Evaluation

Alice (circuit generator)

Bob (circuit evaluator)

Picks $a \in \{0,1\}^s$

Agree on
$f(a,b) \to x$

Picks $b \in \{0,1\}^t$

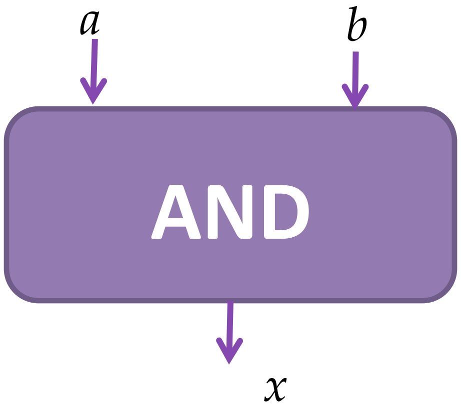
Garbled Circuit Protocol

Outputs $x = f(a,b)$
without revealing a
to Bob or b to Alice.

Andrew Yao, 1982/1986

Regular Logic

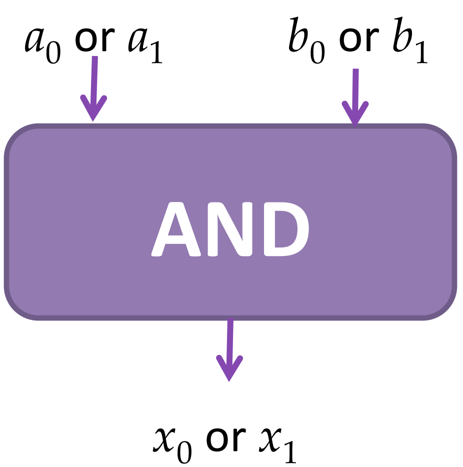
Inputs		Output
a	*b*	*x*
0	0	0
0	1	0
1	0	0
1	1	1



Computing with Meaningless Values?

Inputs		Output
a	b	x
a_0	b_0	x_0
a_0	b_1	x_0
a_1	b_0	x_0
a_1	b_1	x_1

a_i, b_i, x_i are **random** values, chosen by the **circuit generator** but **meaningless** to the **circuit evaluator**.



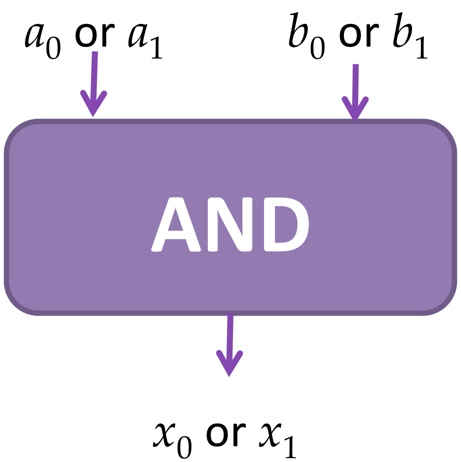
Computing with Garbled Tables

Inputs		Output
a	b	x
a_0	b_0	$Enc_{a0,b0}(x_0)$
a_0	b_1	$Enc_{a0,b1}(x_0)$
a_1	b_0	$Enc_{a1,b0}(x_0)$
a_1	b_1	$Enc_{a1,b1}(x_1)$

Bob can only decrypt **one** of these!

Random Permutation

Garbled And Gate
$Enc_{a0,\ b1}(x_0)$
$Enc_{a1,b1}(x_1)$
$Enc_{a1,b0}(x_0)$
$Enc_{a0,b0}(x_0)$



Garbled Circuit Protocol

Alice (circuit generator)

Bob (circuit evaluator)

Creates random keys: $a_0, a_1, b_0, b_1, x_0, x_1$

Garbled Gate
$Enc_{a0,\ b1}(x_0)$
$Enc_{a1,b1}(x_1)$
$Enc_{a1,b0}(x_0)$
$Enc_{a0,b0}(x_0)$

a_0

Sends a_i to Bob based on her input value

How does the Bob learn his own input wires?

Primitive: **Oblivious Transfer**

Alice

Knows b_0, b_1

Bob

Picks $i \in \{0,1\}$

Oblivious Transfer Protocol

Learns nothing

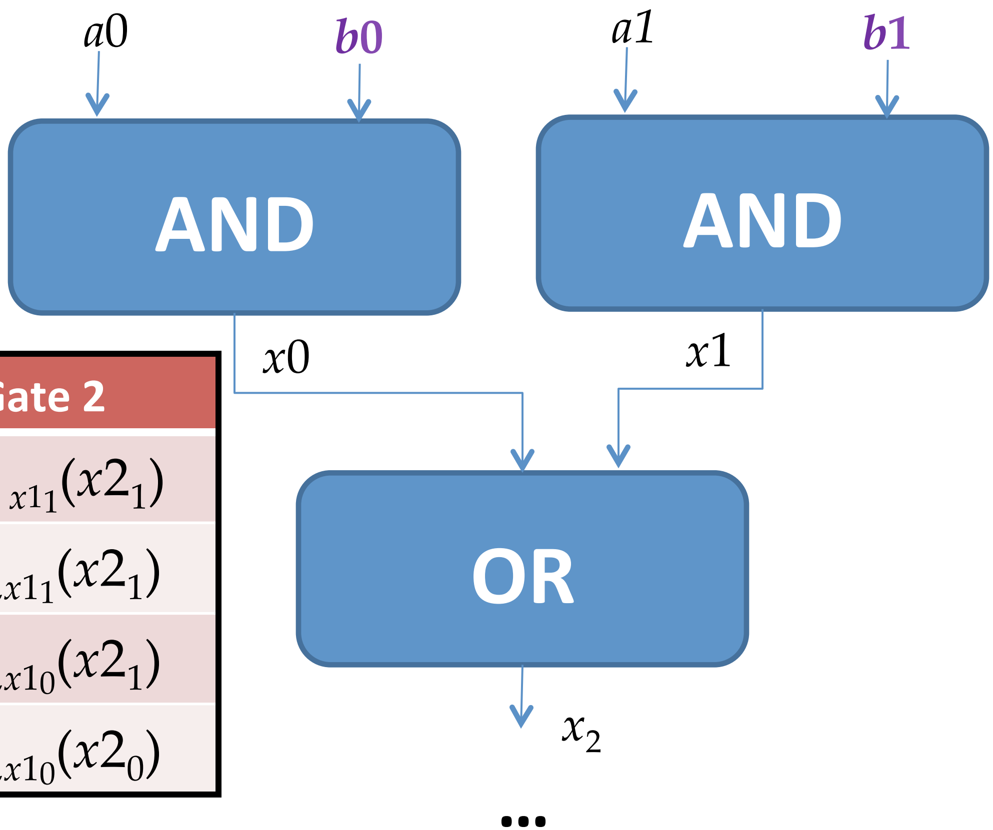
Learns b_i (only)

Oblivious: Alice doesn't learn which secret Bob obtains
Transfer: Bob learns one of Alice's secrets

Rabin, 1981; Even, Goldreich, and Lempel, 1985; many subsequent papers

Chaining Garbled Circuits

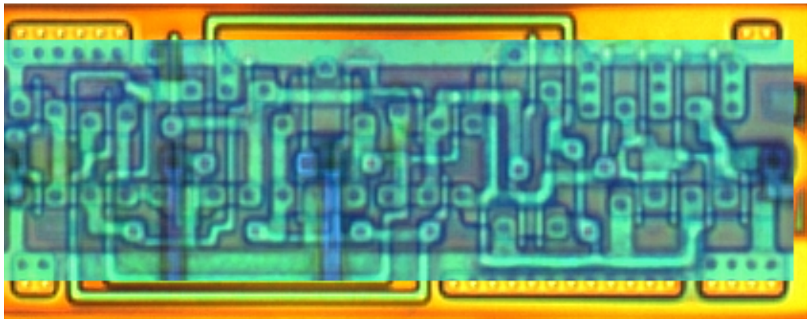
And Gate 1
$Enc_{a1_0,\, b1_1}(x1_0)$
$Enc_{a1_1, b1_1}(x1_1)$
$Enc_{a1_1, b1_0}(x1_0)$
$Enc_{a1_0, b1_0}(x1_0)$



Or Gate 2
$Enc_{x0_0,\, x1_1}(x2_1)$
$Enc_{x0_1, x1_1}(x2_1)$
$Enc_{x0_1, x1_0}(x2_1)$
$Enc_{x0_0, x1_0}(x2_0)$

We can do ***any*** computation privately this way!

Building Computing Systems



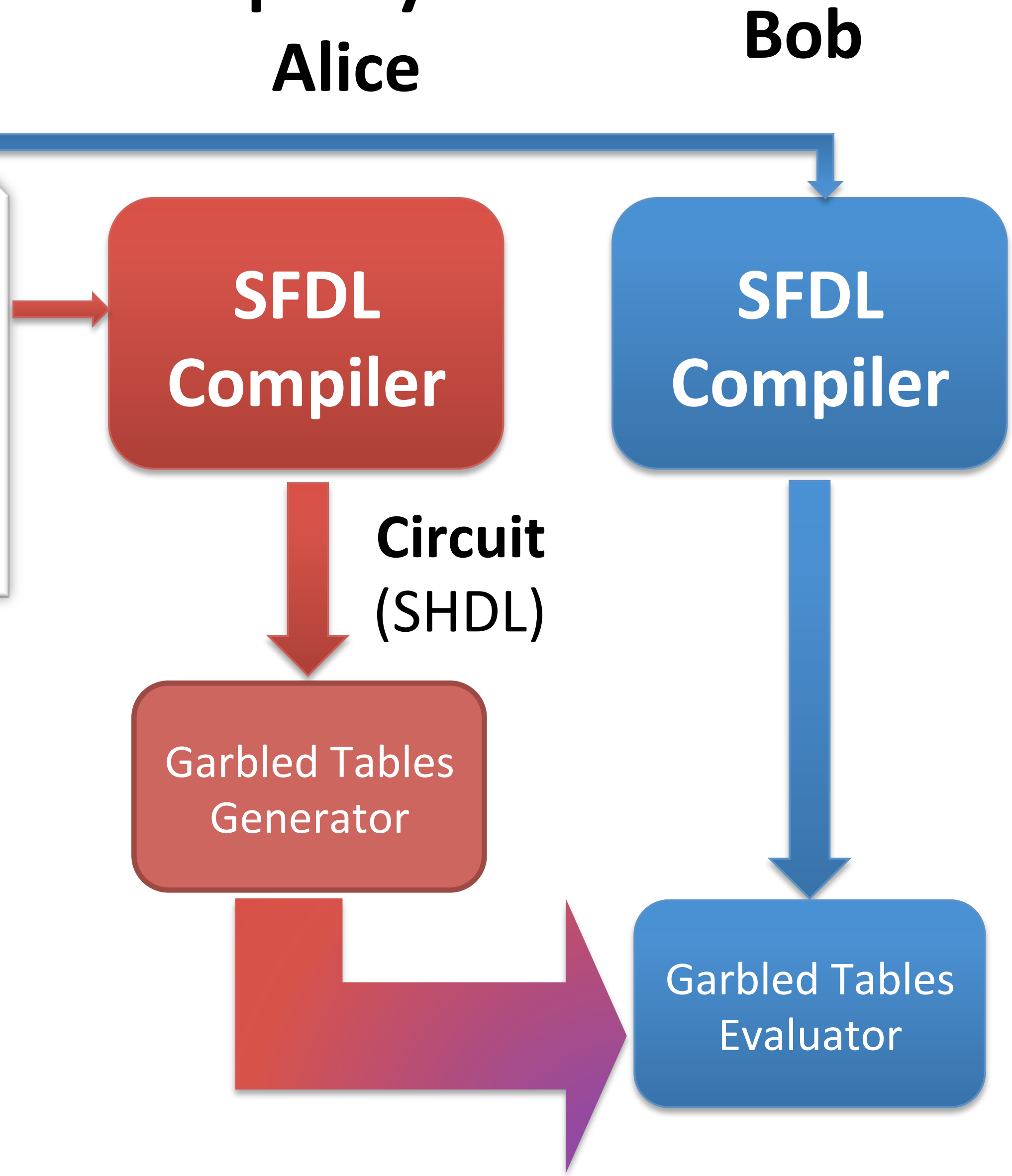
$Enc_{x0_0,\ x1_1}(x2_1)$
$Enc_{x0_1,x1_1}(x2_1)$
$Enc_{x0_1,x1_0}(x2_1)$
$Enc_{x0_0,x1_0}(x2_0)$

Digital Electronic Circuits	Garbled Circuits
Operate on **known data**	Operate on **encrypted wire labels**
One-bit logical operation requires moving a few electrons a few nanometers (hundreds of Billions per second)	One-bit logical operation requires performing (up to) 4 encryption operations: **very slow execution**
Reuse is great!	Reuse is not allowed for privacy: **huge circuits needed**

Fairplay

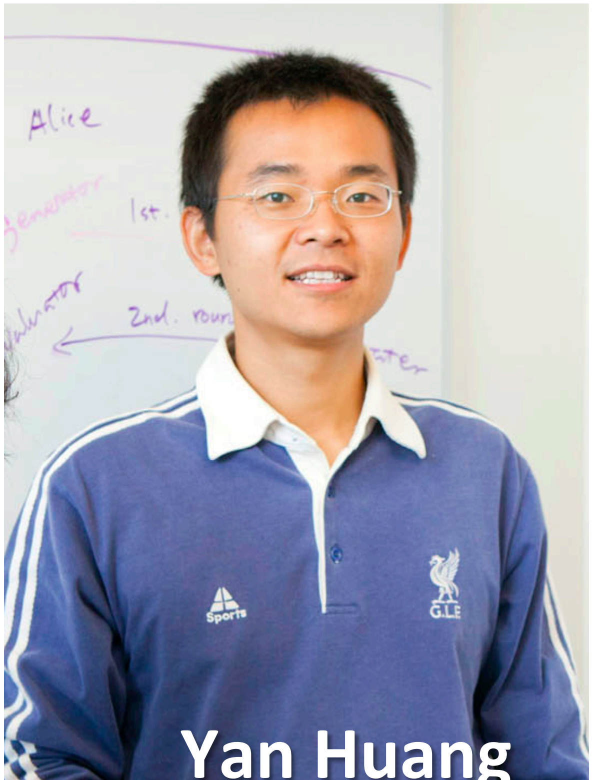
```
program Millionaires {
  type int = Int<4>;  // 4-bit integer
  type AliceInput = int;
  type BobInput = int;
  type AliceOutput = Boolean;
  type BobOutput = Boolean;
  type Output = struct {
       AliceOutput alice, BobOutput bob};
  type Input = struct {
       AliceInput alice, BobInput bob};

  function Output out(Input inp) {
    out.alice = inp.alice > inp.bob;
    out.bob = inp.bob > inp.alice;
```

SFDL Program



Dahlia Malkhi, Noam Nisan, Benny Pinkas and Yaron Sella [USENIX Sec 2004]

Faster Circuit Execution



Yan Huang

(UVa PhD ~~Student~~)
Graduate

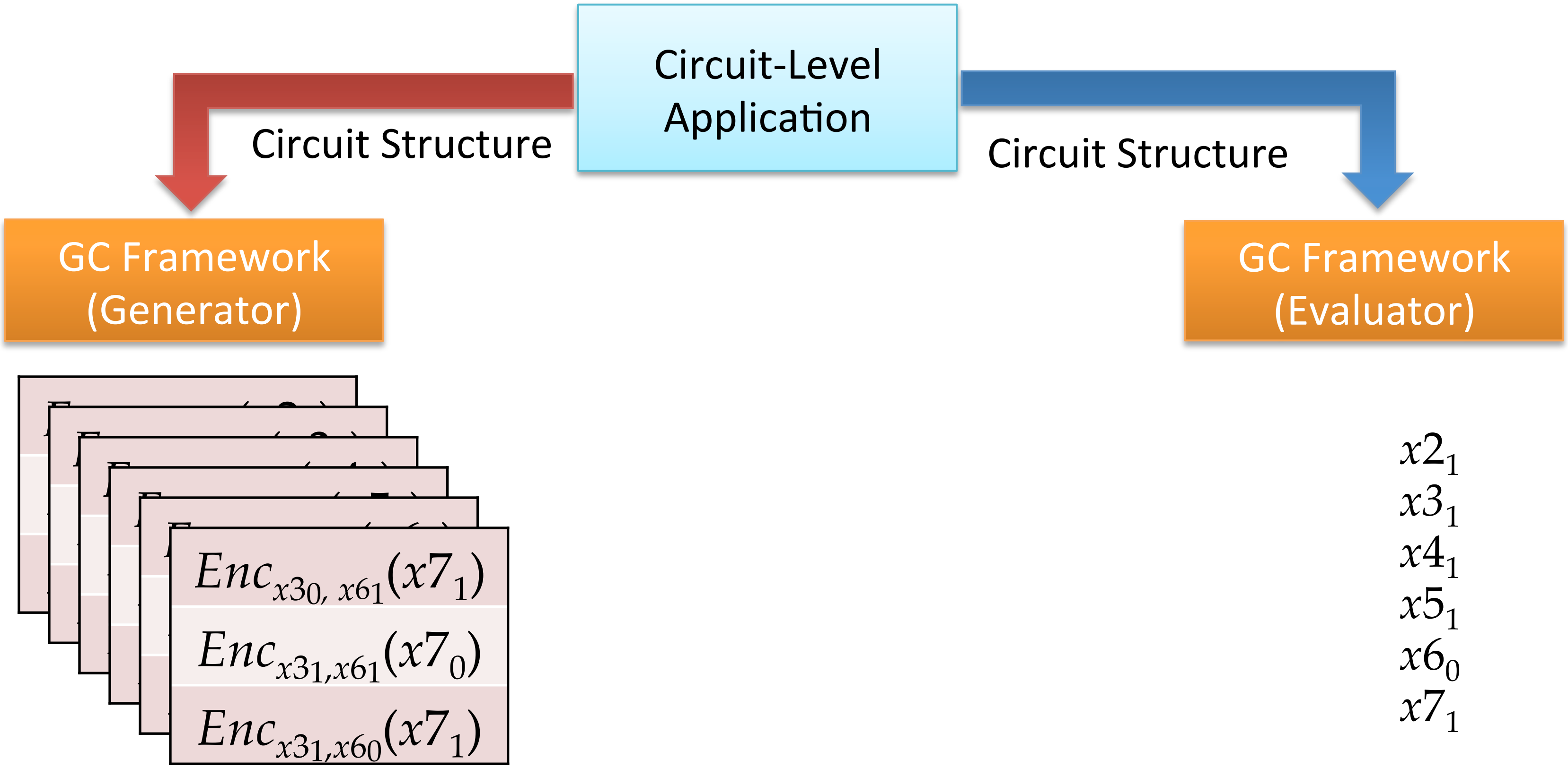
Pipelined Execution

Optimized Circuit Library

Partial Evaluation

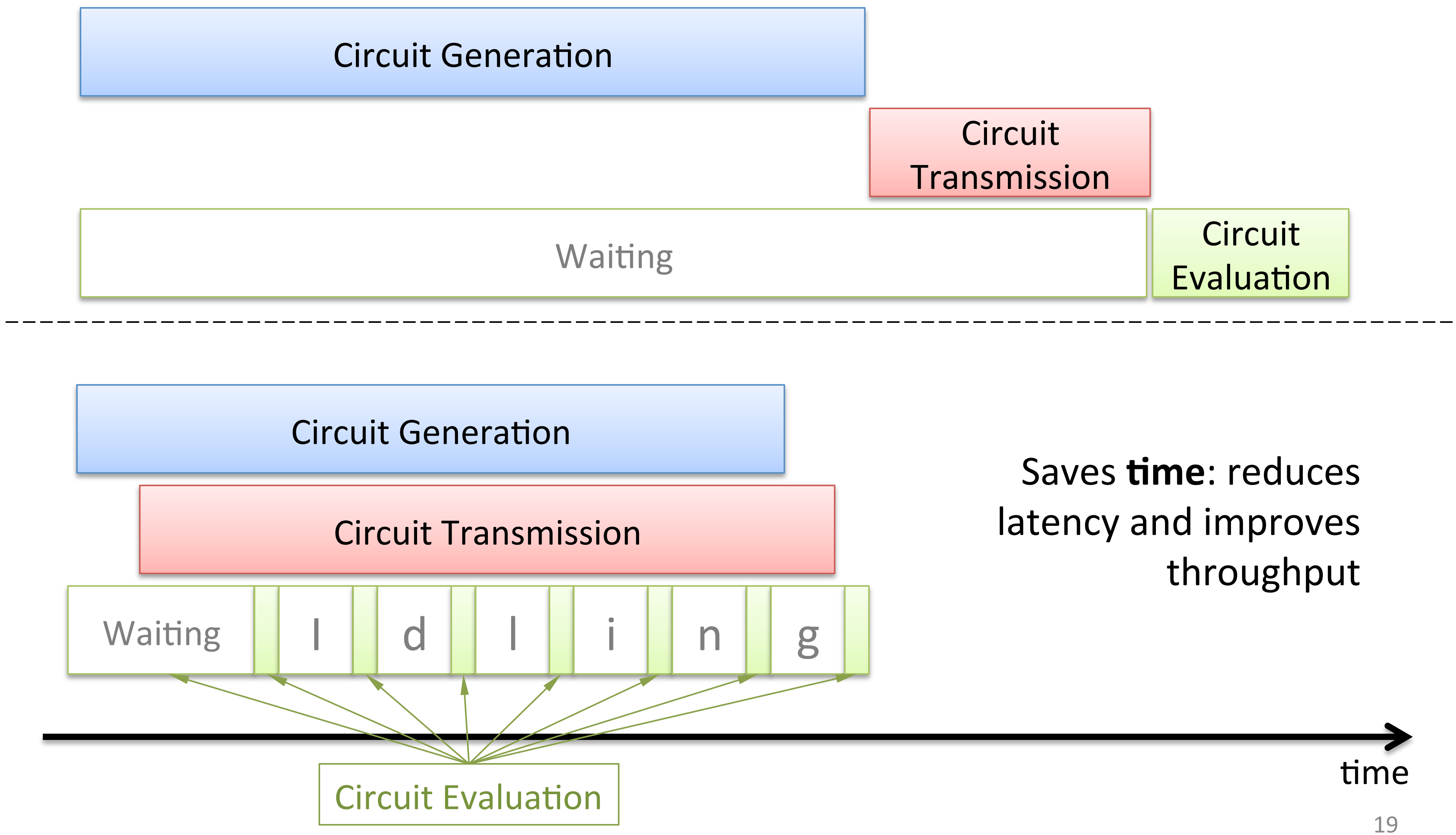
Yan Huang, David Evans, Jonathan Katz, and Lior Malka. *Faster Secure Two-Party Computation Using Garbled Circuits*. **USENIX Security 2011**.

Pipelined Execution



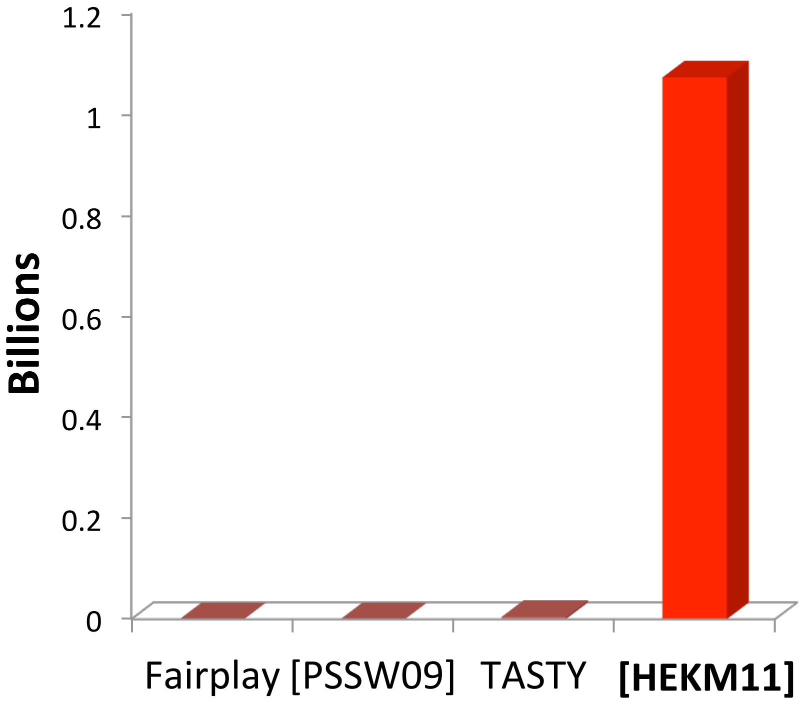
Saves memory: never need to keep whole circuit in memory

Pipelining



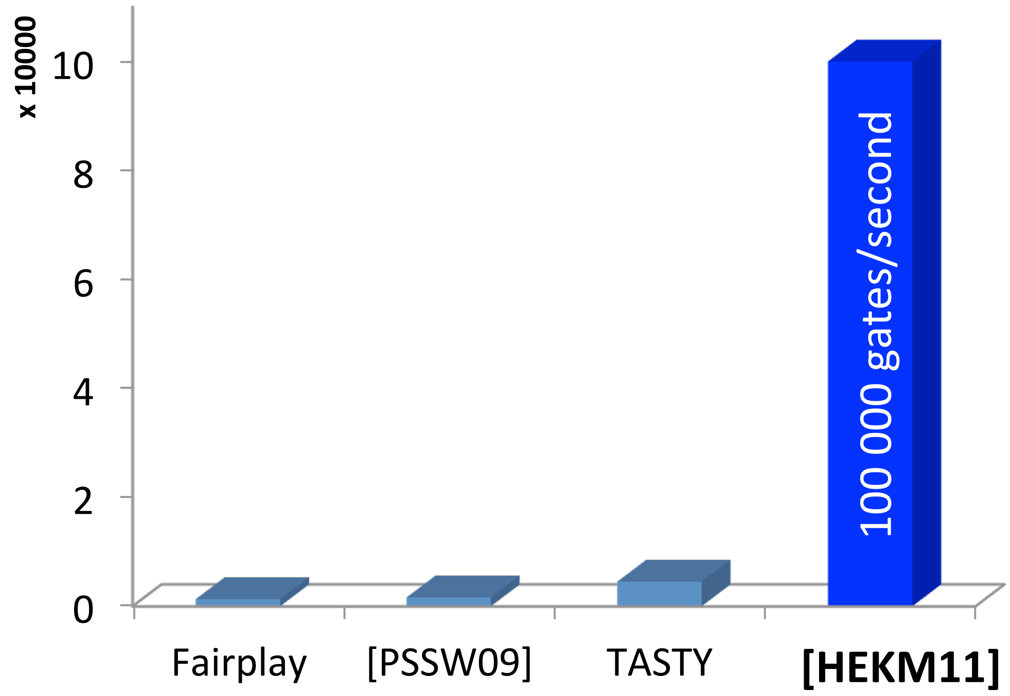
Saves **time**: reduces latency and improves throughput

Results



Scalability

(billions of gates)



Performance

(10,000x non-free gates per second)

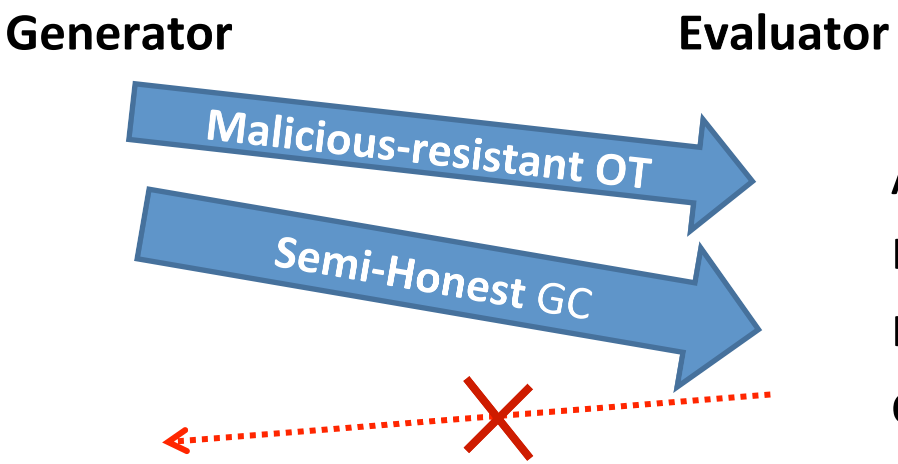
Semi-Honest is Half-Way There

Privacy

Nothing is revealed other than the output

Correctness

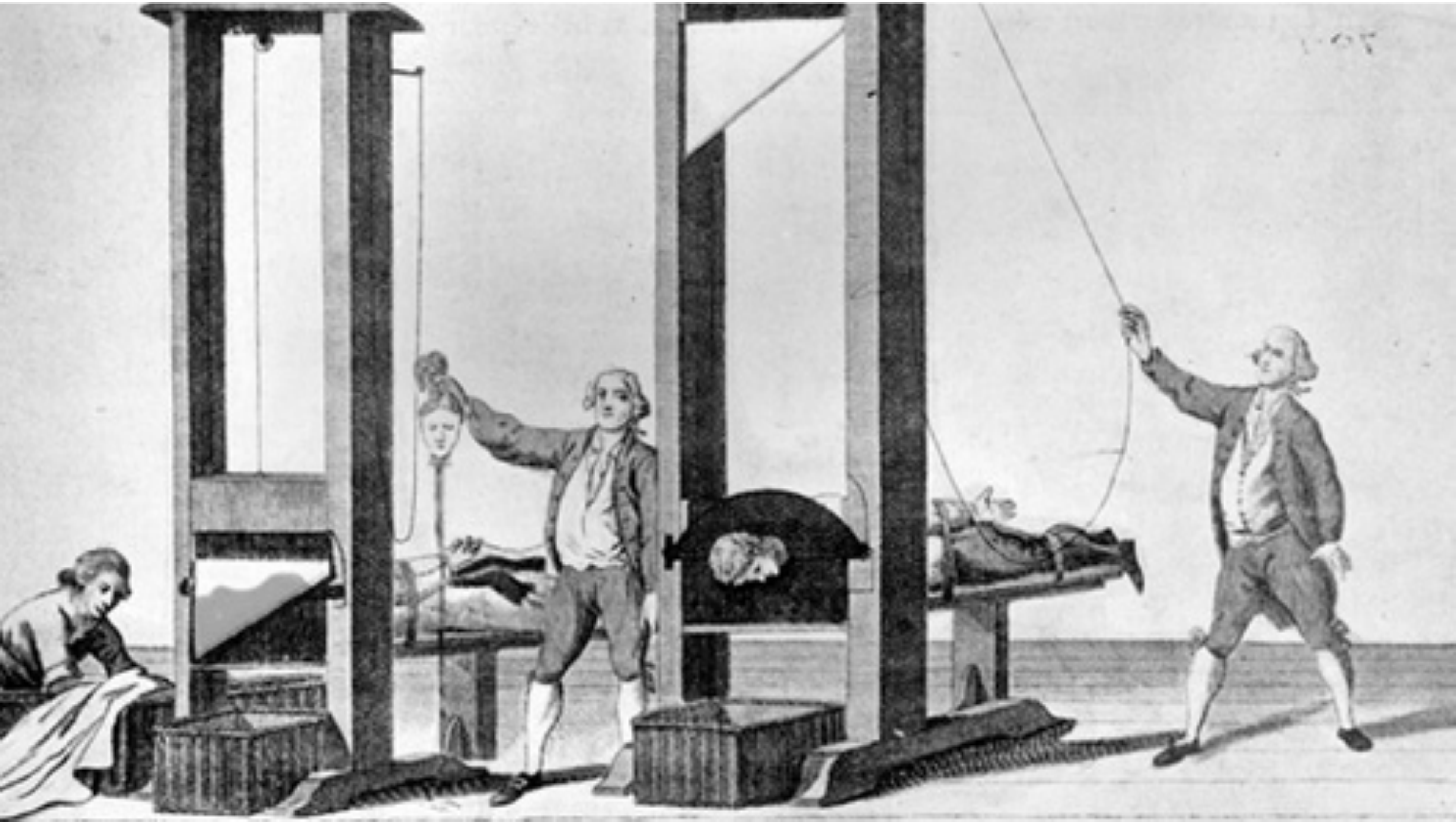
The output of the protocol is indeed $f(x,y)$



As long as evaluator doesn't send result back, and a malicious-resistant OT is used, **privacy** for evaluator is guaranteed.

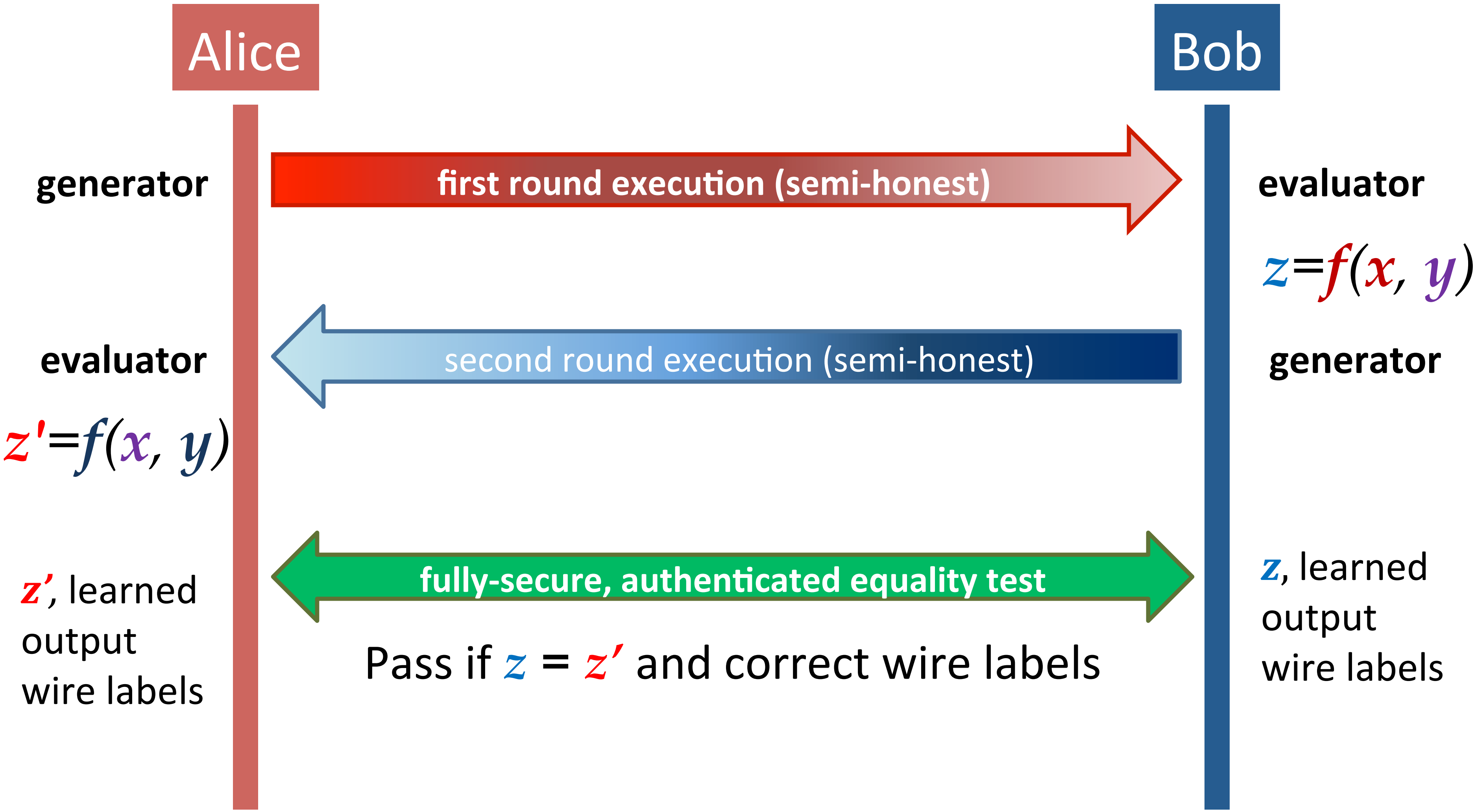
How can we get both correctness, and maintain privacy while giving both parties result?

Dual Execution Protocols



Yan Huang, Jonathan Katz, and David Evans. *Quid-Pro-Quo-tocols: Strengthening Semi-Honest Protocols with Dual Execution.* **IEEE Security and Privacy (Oakland) 2012.**

Dual Execution Protocol



[Mohassel and Franklin, PKC'06]

Security Properties

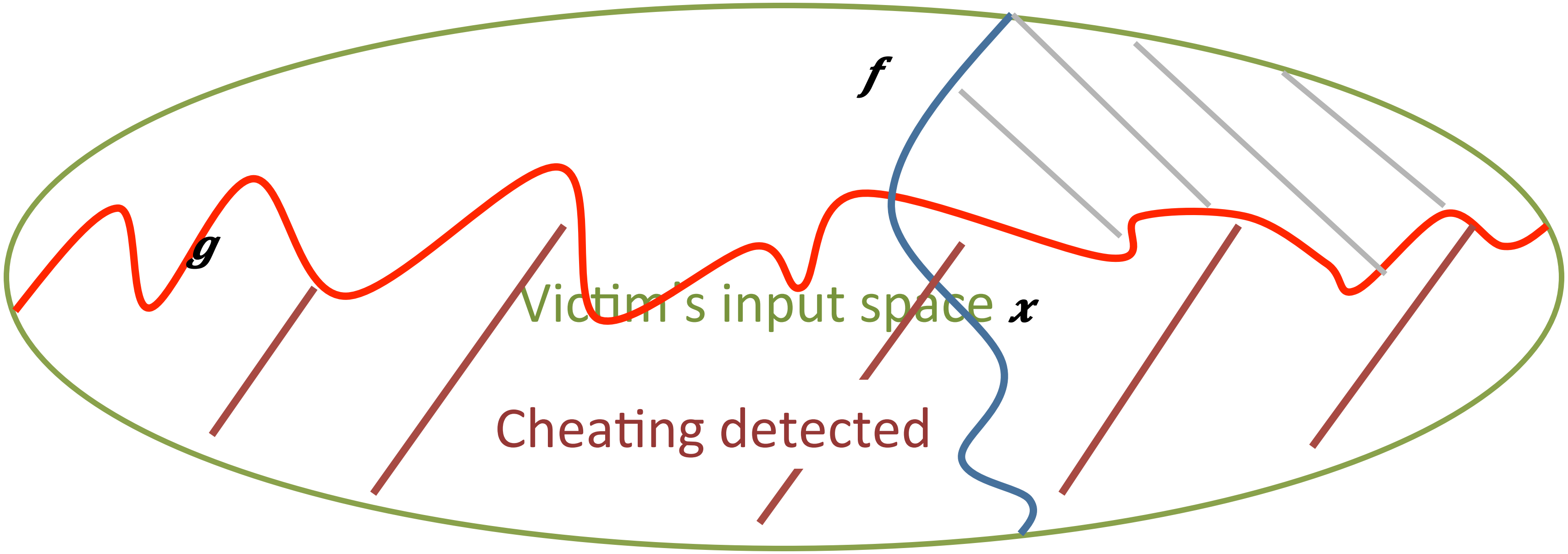
Correctness: guaranteed by authenticated, secure equality test

Privacy: Leaks **one** (extra) **bit** on average

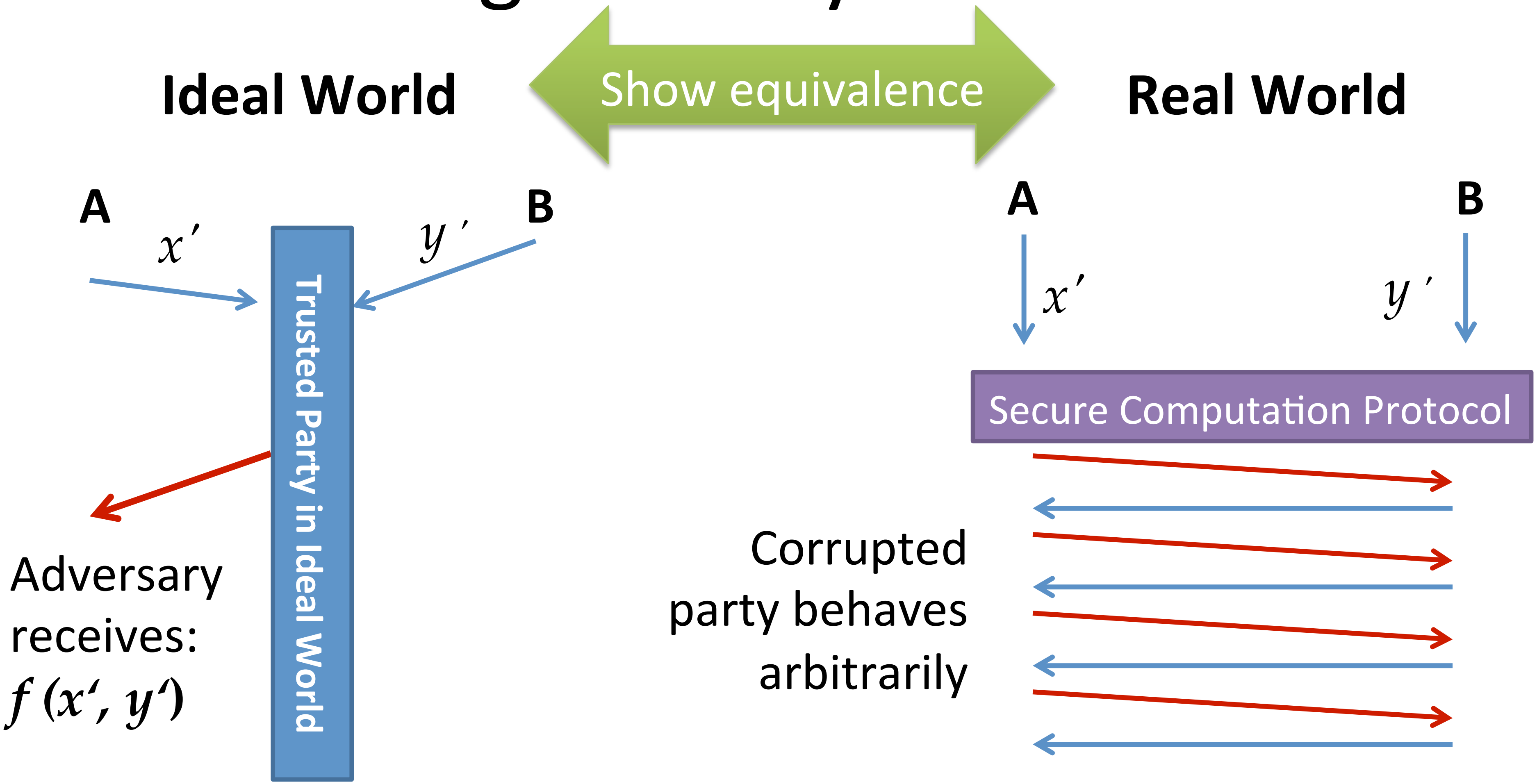
adversarial circuit generator provides a circuit that fails on ½ of inputs

Malicious generator can **decrease likelihood of being caught**, and **increase information leaked when caught** (but decreases average information leaked): at extreme, circuit fails on just one input

1-bit Leak



Proving Security: Malicious

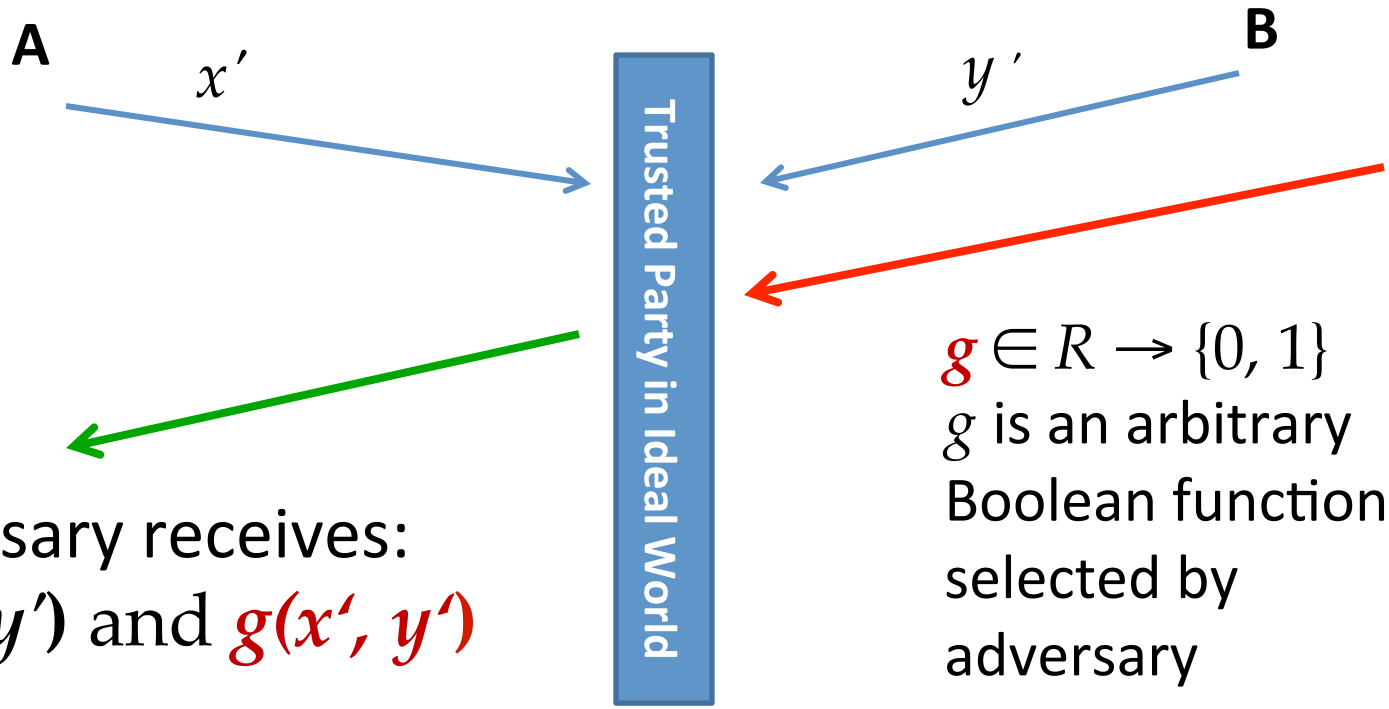


Standard Malicious Model: can't prove this for Dual Execution

Proof of Security: One-Bit Leakage

Ideal World

Controlled by malicious $\mathcal{A}$



$g \in R \rightarrow \{0, 1\}$

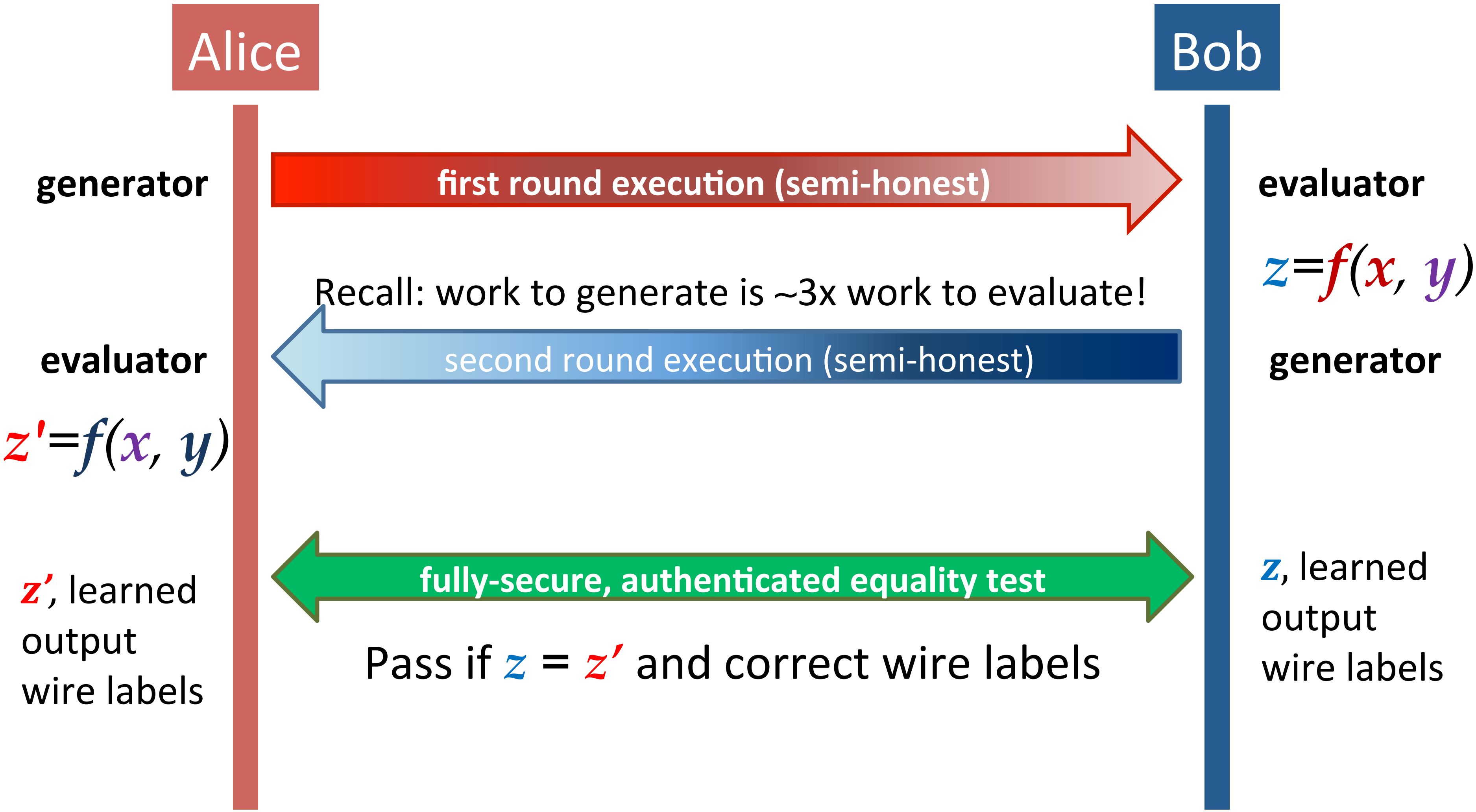
g is an arbitrary Boolean function selected by adversary

Adversary receives:

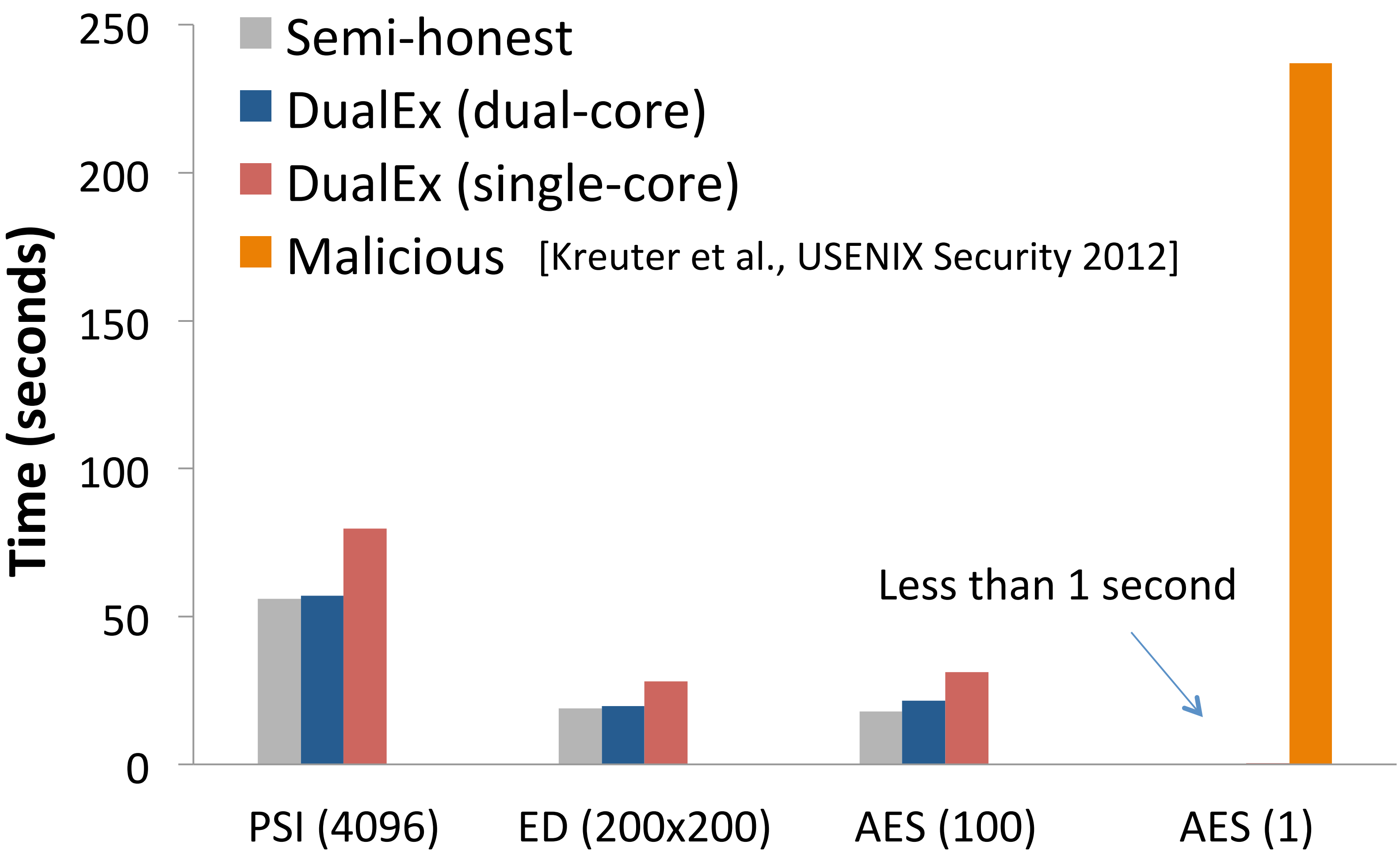
$f(x', y')$ and $g(x', y')$

Can prove equivalence to this for Dual Execution protocols

Implementation



Performance



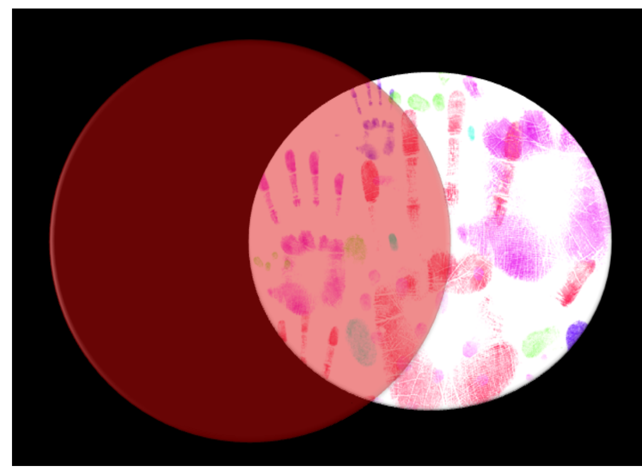
Circuits of arbitrary size can be done this way

Applications

Privacy-Preserving
Biometric Matching

Private
Personal
Genomics

Private AES
Encryption



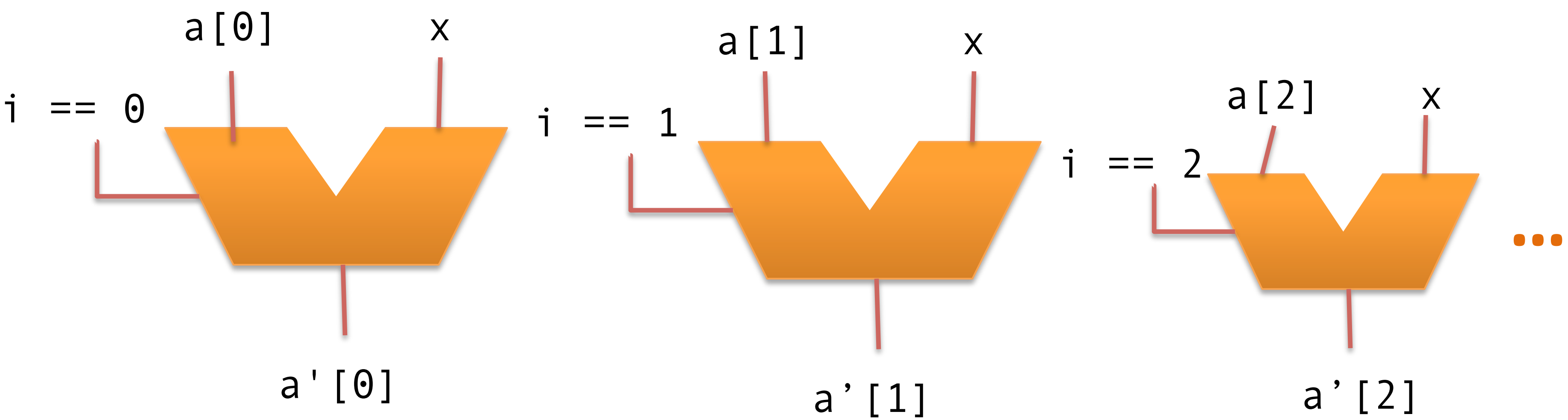
Private Set Intersection

	Problem	Best Previous Result	Our Result	Speedup
NDSS 2012	**Private Set Intersection** (contact matching, common disease carrier)	Competitive with best custom protocols, scales to millions of 32-bit elements		
USENIX Security 2011	**Hamming Distance** (Face Recognition)	213s [SCiFI, 2010]	**0.051s**	**4176**
USENIX Security 2011	**Levenshtein Distance** (genome, text comparison) – two 200-character inputs	534s [Jha+, 2008]	**18.4s**	**29**
USENIX Security 2011	**Smith-Waterman** (genome alignment) – two 60-nucleotide sequences	[Not Implementable]	**447s**	**-**
USENIX Security 2011	**AES Encryption**	3.3s [Henecka, 2010]	**0.2s**	**16.5**
NDSS 2011	**Fingerprint Matching** (1024-entry database, 640x8bit vectors)	~83s [Barni, 2010]	**18s**	**4.6**

Crazy Things in Typical Code

```
a[i] = x
```

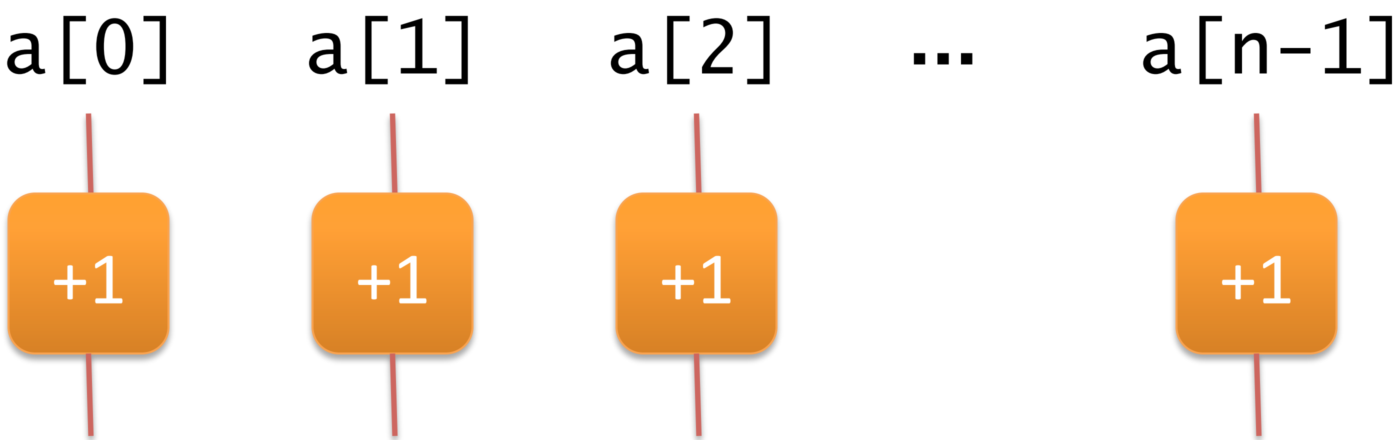
Circuit for Array Update



a[i] = x

Easy (and Common) Case

```
for (i = 0; i < n; i++)
    a[i] += 1
```



Circuit Structures

Design circuits to support typical **data structures** efficiently

Non-trivial access patterns, but **patterns nonetheless**

Main opportunities:

Locality and Batching

(UVa PhD Student)

Samee Zahur and David Evans. *Circuit Structures for Improving Efficiency of Security & Privacy Tools.* **IEEE Security and Privacy (Oakland) 2013.**

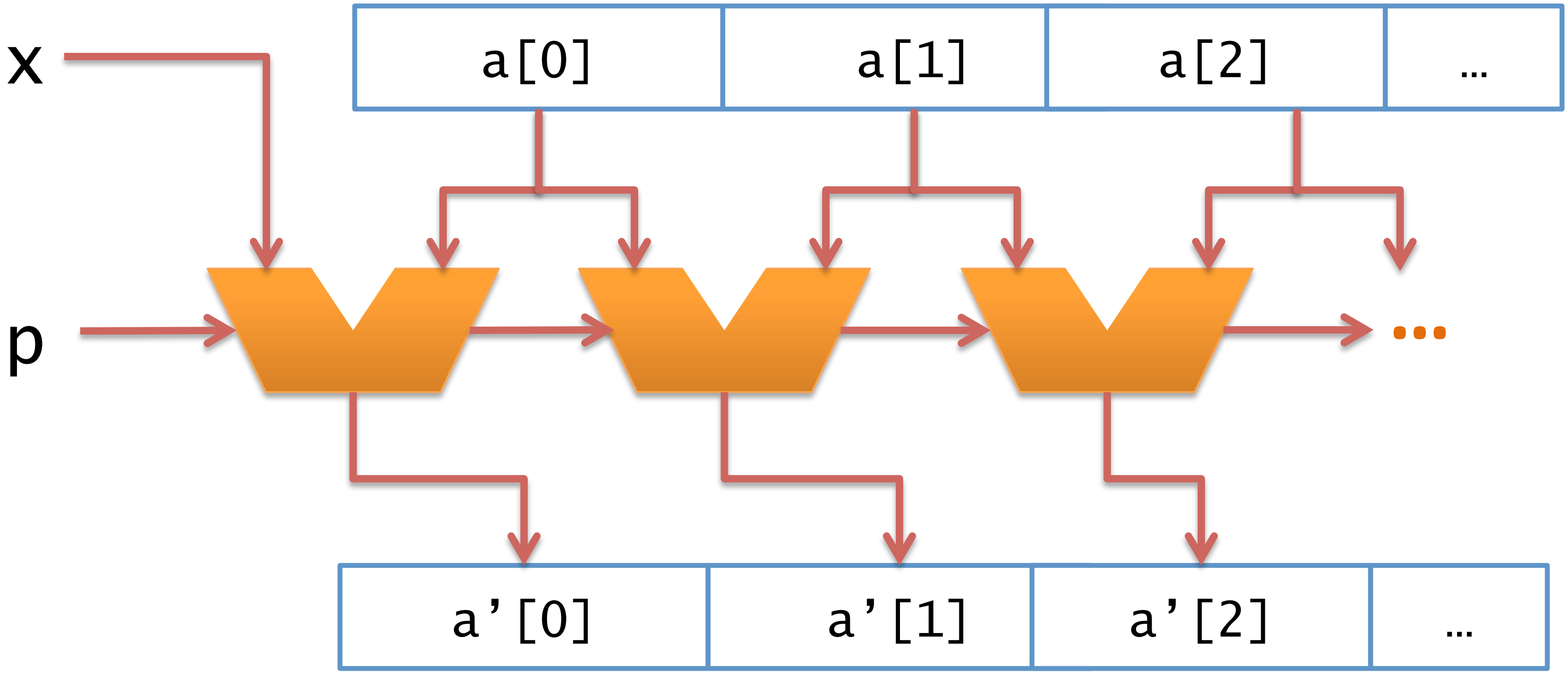
Locality: Stacks and Queues

```
if (x != 0)
   a[i] += 1
   if (a[i] > 10)
      i += 1
   a[i] = 5
```

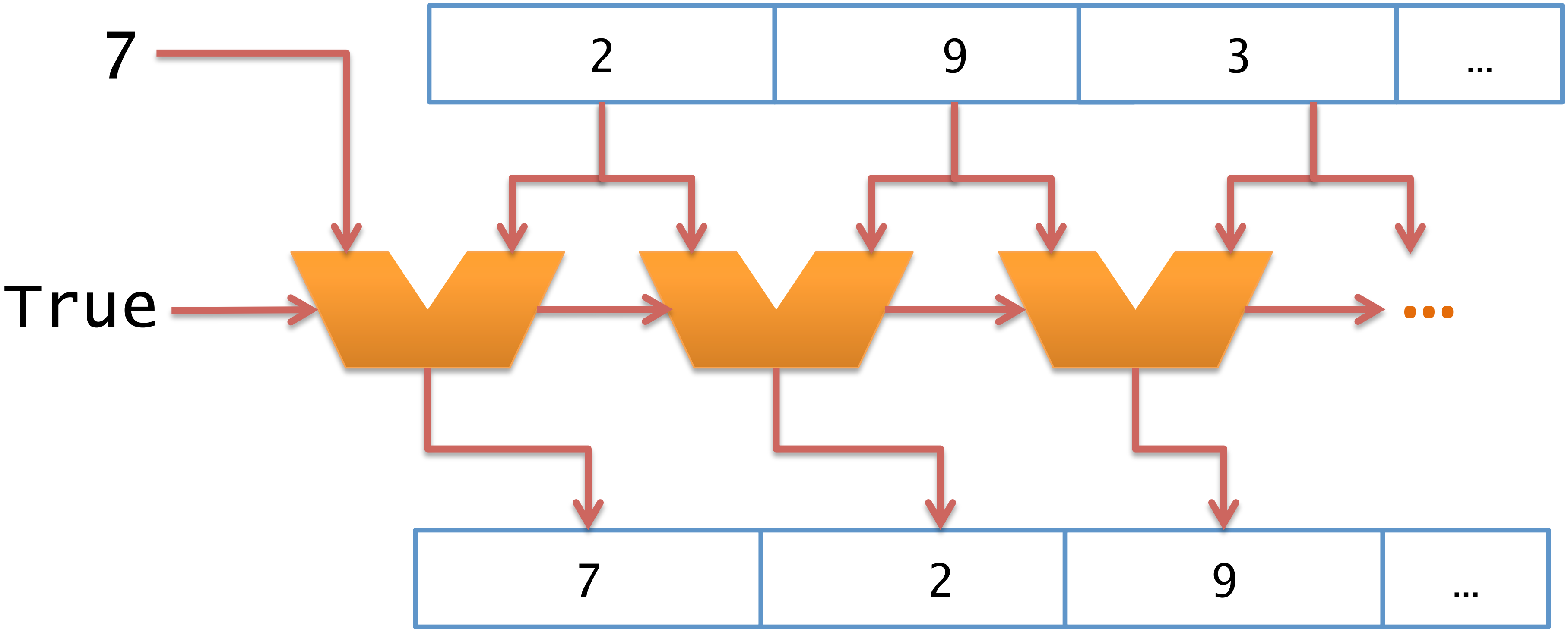
Data-oblivious code
No branching allowed

```
t := a.top() + 1
a.cond_update(x != 0, t)
a.cond_push(x != 0 && t > 10, *)
a.cond_update(x != 0, 5)
```

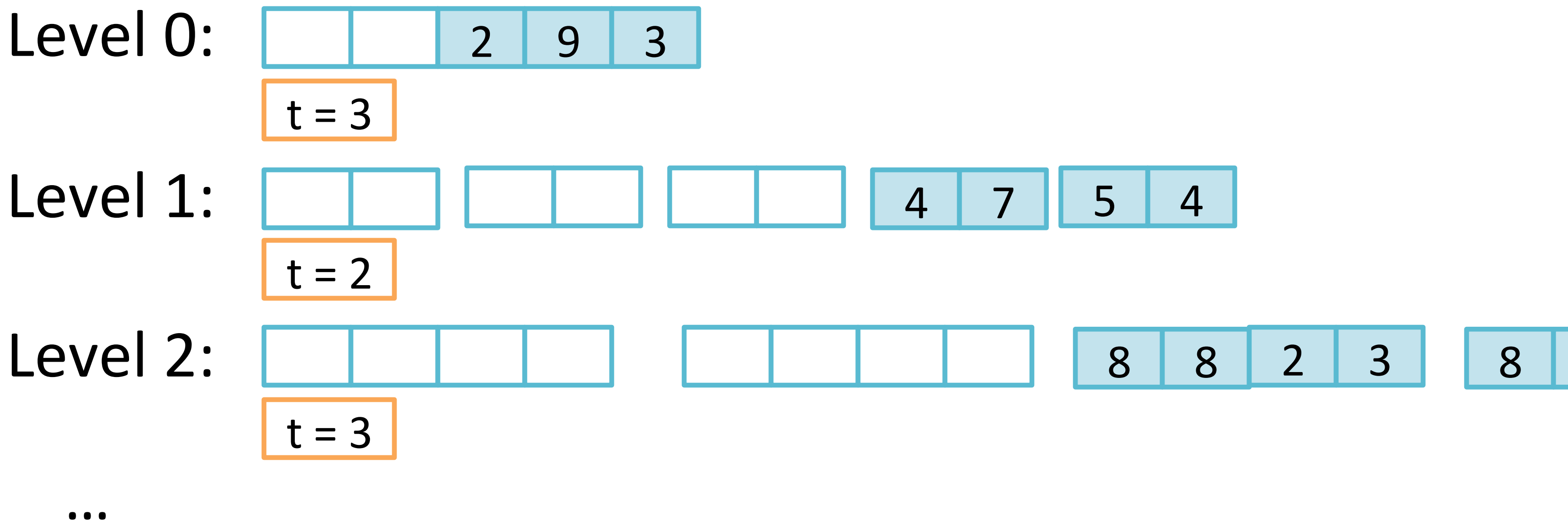
Naïve Conditional Push



Naïve Conditional Push

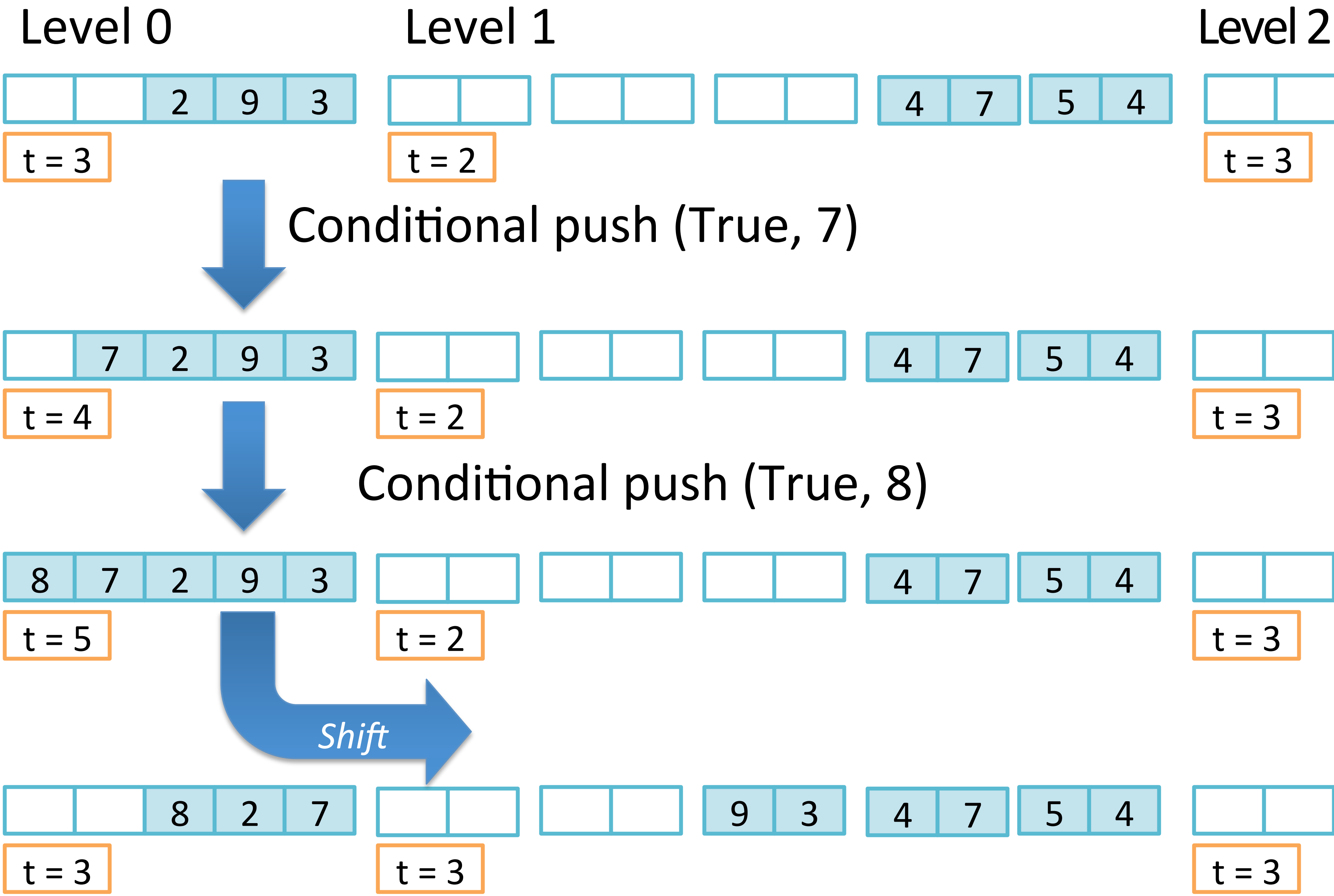


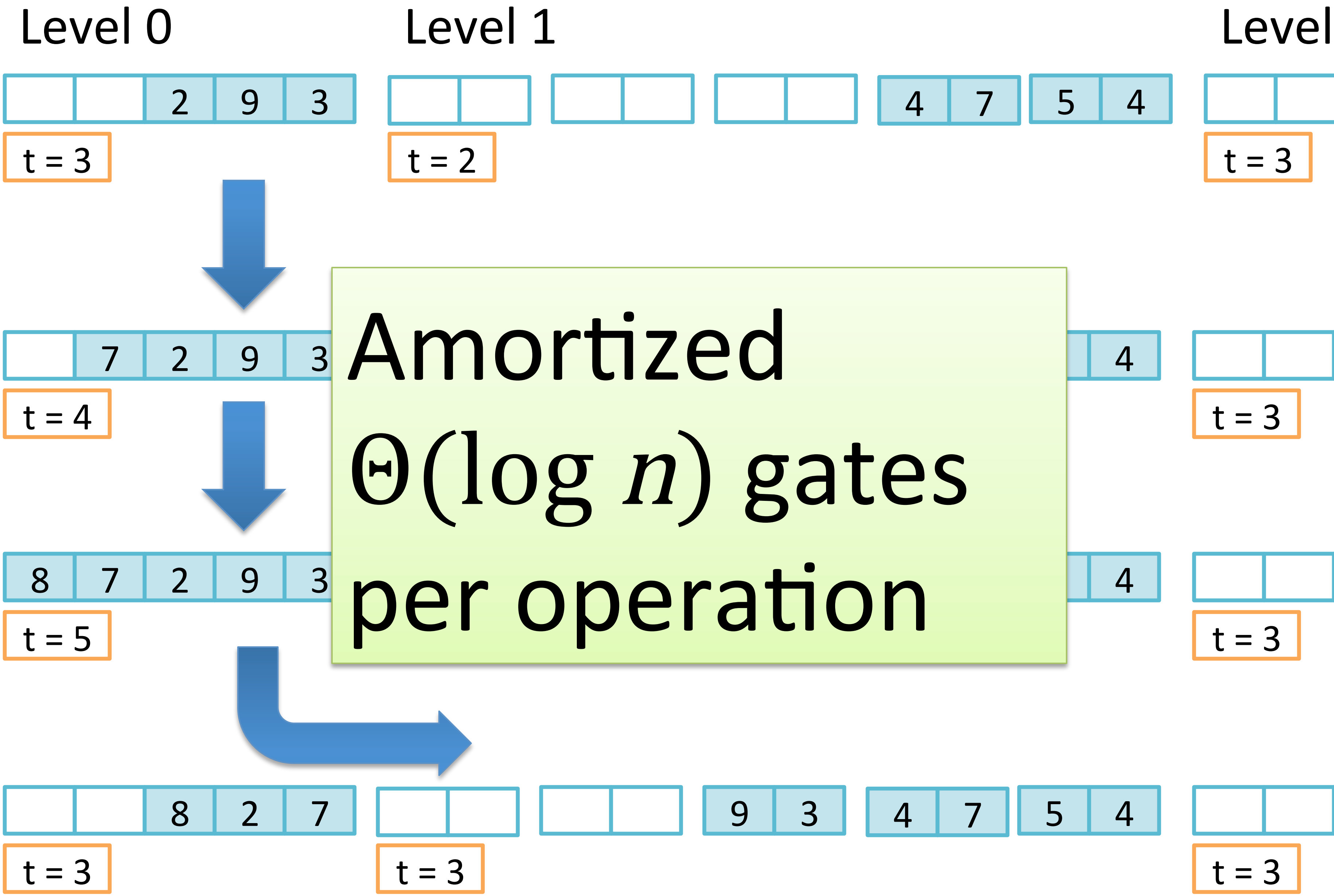
More Efficient Stack



Block size = 2^{level}

Each level has 5 blocks, at least 2 full and 2 empty


Level 0
Level 1
Level 2
2 9 3
4 7
5 4
t = 3
t = 2
t = 3
Conditional push (True, 7)
7 2 9 3
4 7
5 4
t = 4
t = 2
t = 3
Conditional push (True, 8)
8 7 2 9 3
4 7
5 4
t = 5
t = 2
t = 3
Shift
8 2 7
9 3
4 7
5 4
t = 3
t = 3
t = 3


Level 0
Level 1
Level
2 9 3
4 7 5 4
t = 3
t = 2
t = 3
7 2 9 3
4
t = 4
t = 3
8 7 2 9 3
4
t = 5
t = 3
8 2 7
9 3 4 7 5 4
t = 3
t = 3
t = 3
Amortized
Θ(log n) gates
per operation

Arbitrary Array Accesses (*Associative Maps*)

```
m[0] = 'A'
m[2] = 'U'
m[9] = 'M'
m[7] = 'R'
m[0] = 'D'
m[9] = 'Y'
m[9] = 'K'
m[7] = 'C'
```

0	2	7	9
'A'			
	'U'		
			'M'
		'R'	
'D'			
			'Y'
			'K'
		'C'	

Execution trace: **indexes and values are private values**

Batching Updates

```
m[0] = 'A'
m[2] = 'U'
m[9] = 'M'
m[7] = 'R'
m[0] = 'D'
m[9] = 'Y'
m[9] = 'K'
m[7] = 'C'
```

Sort by Key

stable sort!

```
m[0] = 'A'
m[0] = 'D'
m[2] = 'U'
m[7] = 'R'
m[7] = 'C'
m[9] = 'M'
m[9] = 'Y'
m[9] = 'K'
```

Batching Updates

```
m[0] = 'A'
m[2] = 'U'
m[9] = 'M'
m[7] = 'R'
m[0] = 'D'
m[9] = 'Y'
m[9] = 'K'
m[7] = 'C'
```

Sort by Key

stable sort!

```
m[0] = 'A'
m[0] = 'D'
m[2] = 'U'
m[7] = 'R'
m[7] = 'C'
m[9] = 'M'
m[9] = 'Y'
m[9] = 'K'
```

Batching Updates

```
'A'
'U'
'M'
'R'
'D'
'Y'
'K'
'C'
```

Sort by Key →

stable sort!

```
m[0] = 'A'
m[0] = 'D'
m[2] = 'U'
m[7] = 'R'
m[7] = 'C'
m[9] = 'M'
m[9] = 'Y'
m[9] = 'K'
```

Compare Adjacent →

```
m[0] = 'A'
m[0] = 'D'
m[2] = 'U'
m[7] = 'R'
m[7] = 'C'
m[9] = 'M'
m[9] = 'Y'
m[9] = 'K'
```

Batching Updates

```
'A'
'U'
'M'
'R'
'D'
'Y'
'K'
'C'
```

Sort by Key

stable sort!

```
m[0] = 'A'
m[0] = 'D'
m[2] = 'U'
m[7] = 'R'
m[7] = 'C'
m[9] = 'M'
m[9] = 'Y'
m[9] = 'K'
```

Compare Adjacent

```
m[0] = 'A'
m[0] = 'D'
m[2] = 'U'
m[7] = 'R'
m[7] = 'C'
m[9] = 'M'
m[9] = 'Y'
m[9] = 'K'
```

Batching Updates

```
'A'
'U'
'M'
'R'
'D'
'Y'
'K'
'C'
```

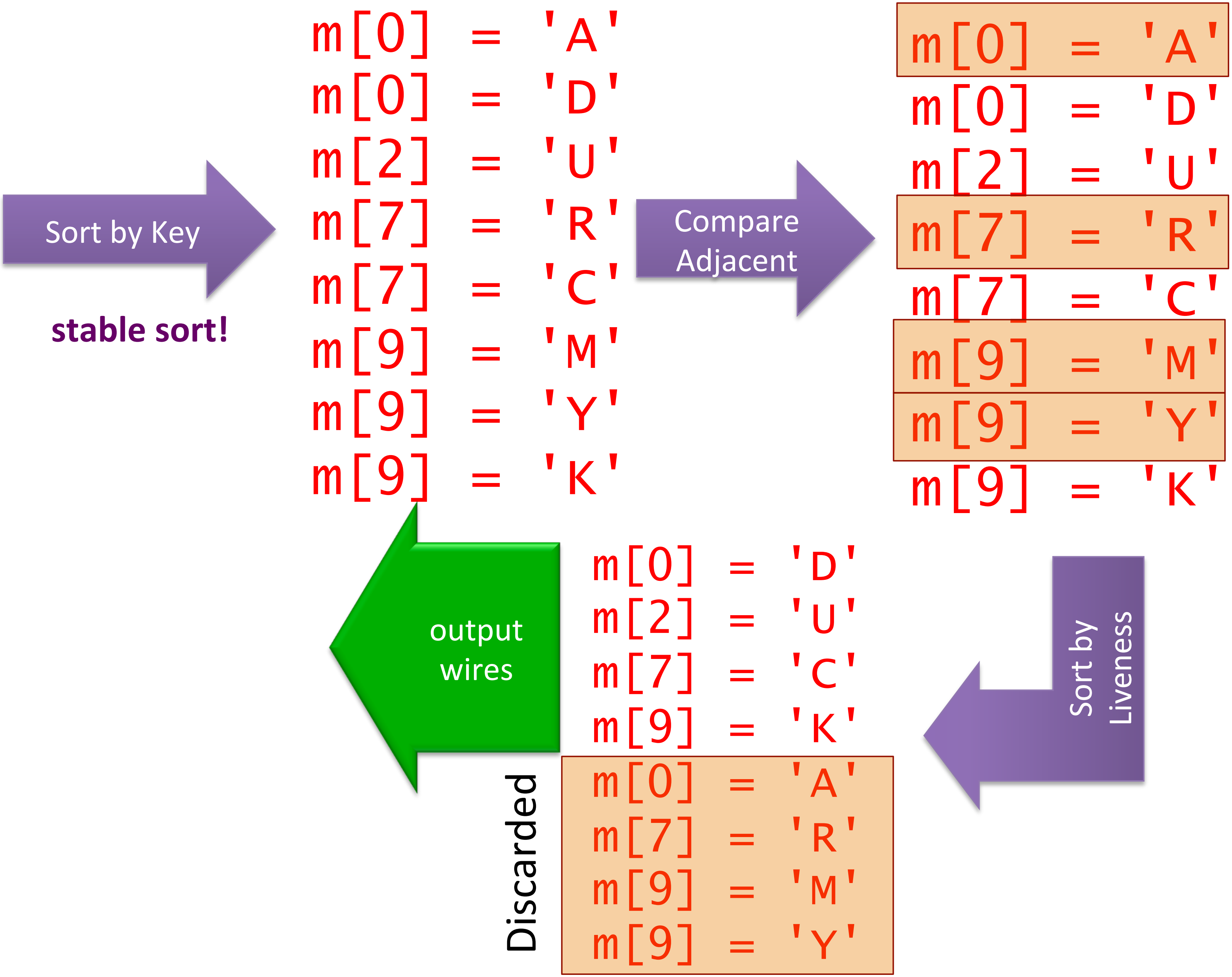
Sort by Key

stable sort!

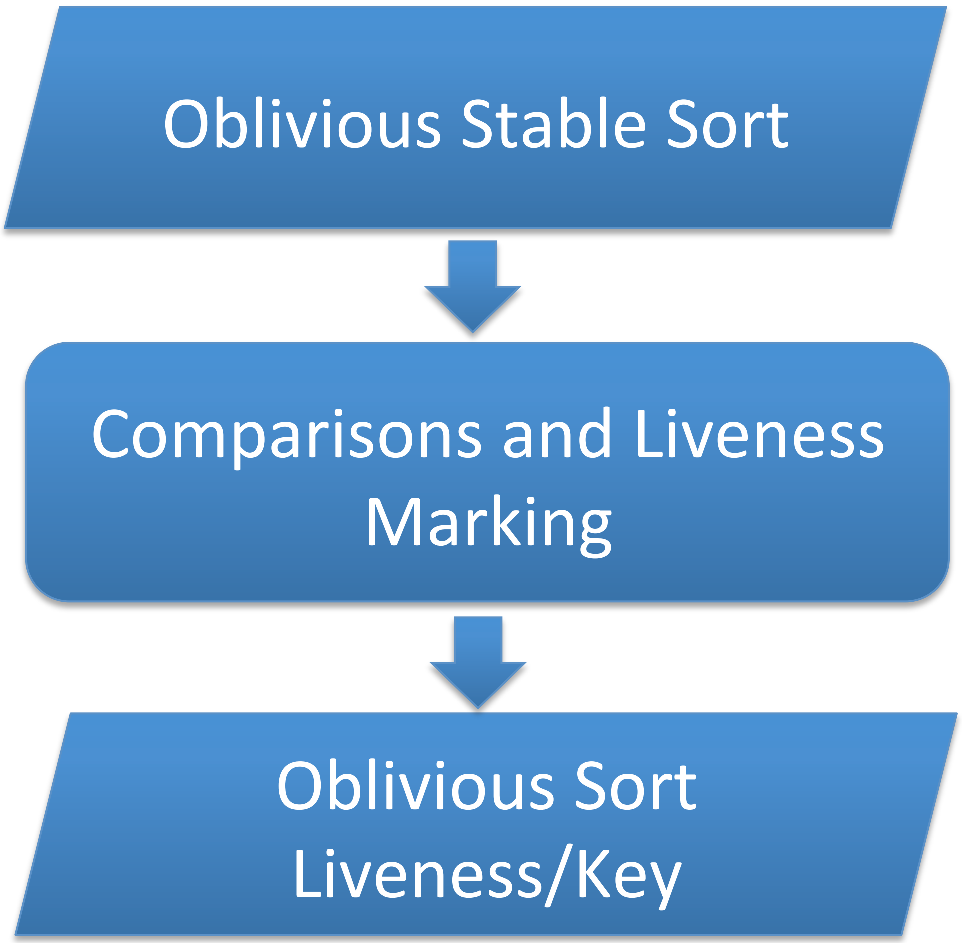
```
m[0] = 'A'
m[0] = 'D'
m[2] = 'U'
m[7] = 'R'
m[7] = 'C'
m[9] = 'M'
m[9] = 'Y'
m[9] = 'K'
```

Compare Adjacent

```
m[0] = 'A'
m[0] = 'D'
m[2] = 'U'
m[7] = 'R'
m[7] = 'C'
m[9] = 'M'
m[9] = 'Y'
m[9] = 'K'
```


Sort by Key
stable sort!
m[0] = 'A'
m[0] = 'D'
m[2] = 'U'
m[7] = 'R'
m[7] = 'C'
m[9] = 'M'
m[9] = 'Y'
m[9] = 'K'
Compare Adjacent
m[0] = 'A'
m[0] = 'D'
m[2] = 'U'
m[7] = 'R'
m[7] = 'C'
m[9] = 'M'
m[9] = 'Y'
m[9] = 'K'
Sort by Liveness
m[0] = 'D'
m[2] = 'U'
m[7] = 'C'
m[9] = 'K'
m[0] = 'A'
m[7] = 'R'
m[9] = 'M'
m[9] = 'Y'
Discarded
output wires

Associative Map Cost



Circuit Size:

$\Theta(n \log n) \times$ comparison cost

$\Theta(n \log^2 n)$

Example Application: **DBScan**

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu. *KDD* 1996

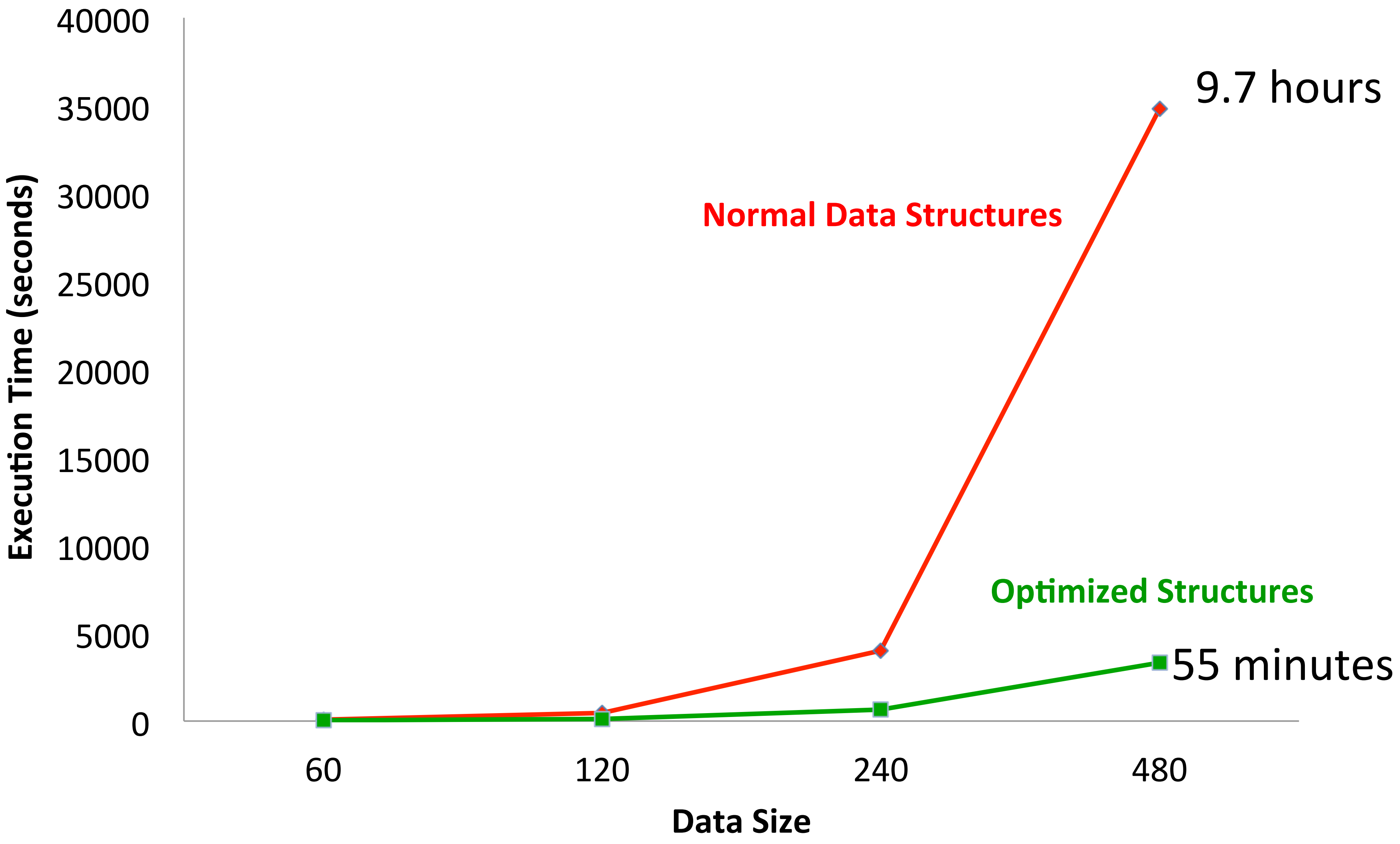
Density-based clustering:
depth-first search to find dense clusters

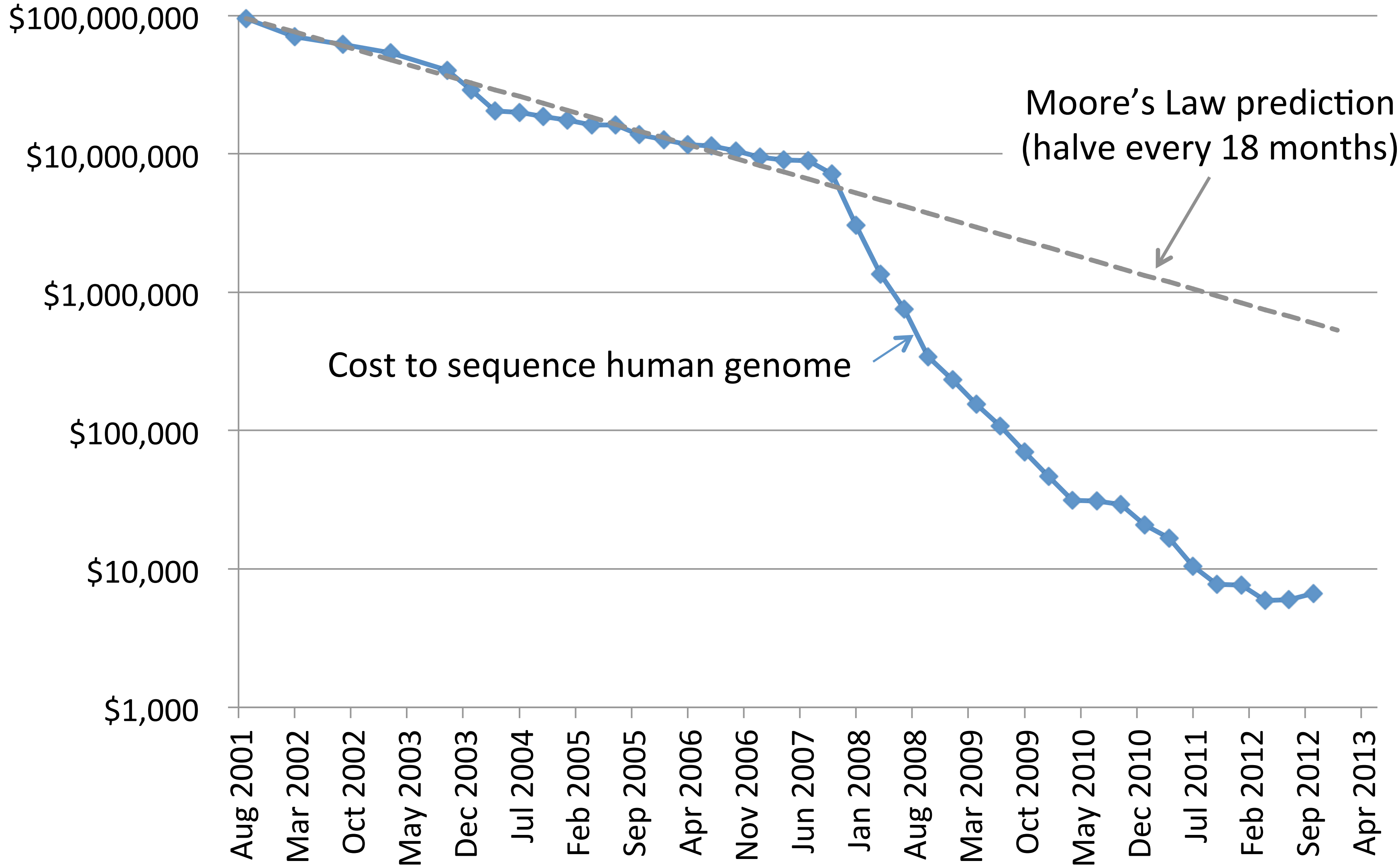
Private Input: *P* – array of points (combines private points from both parties)
Public inputs: *minpts, radius*
Output: cluster number for each point

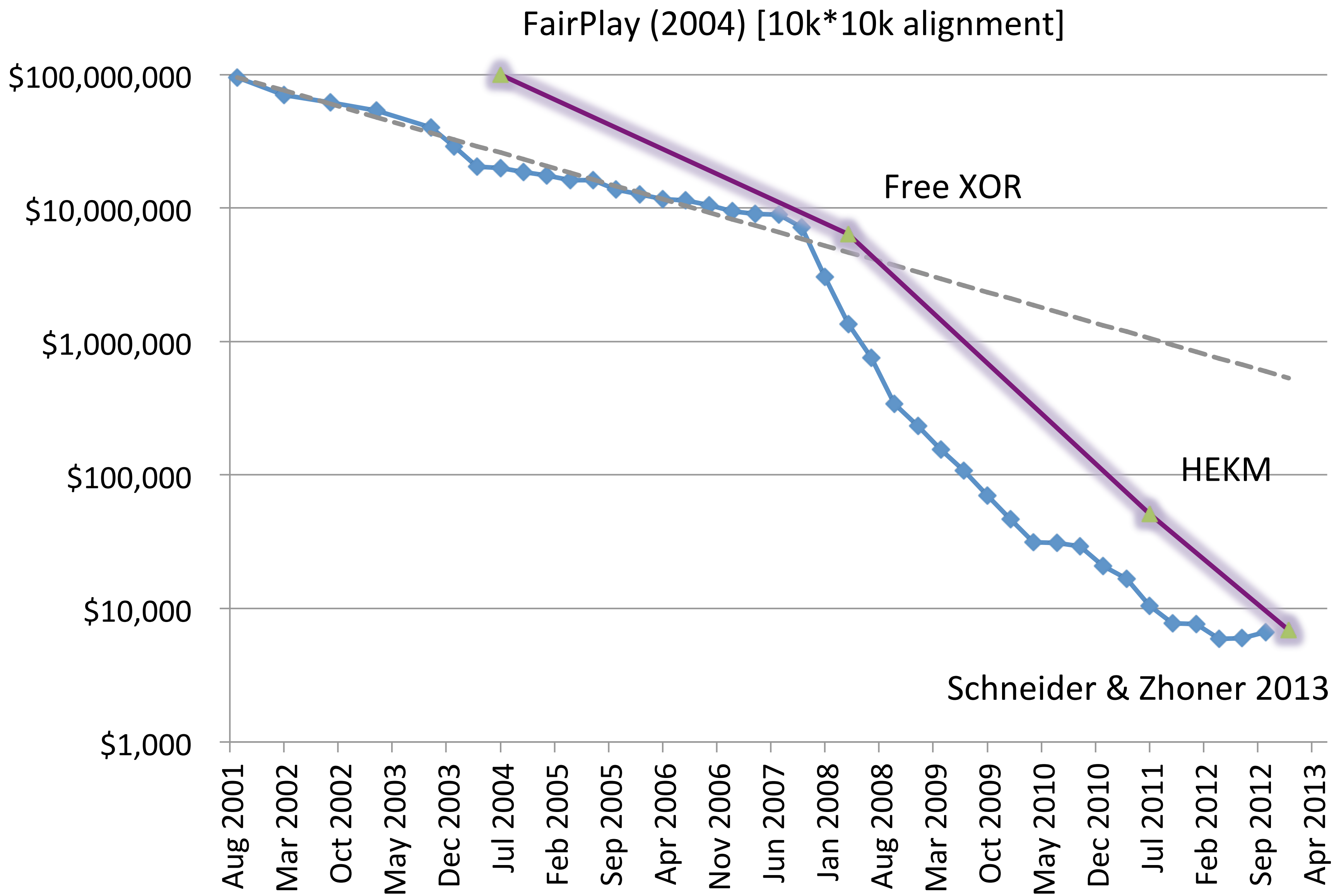
```
n ← |P|
c ← 0
s ← emptyStack
cluster ← [0, 0, ...]
for i ← [1, n] do
    if cluster[i] ≠ 0 then
        continue
    V ← getNeighbors(i, P, minpts, radius)
    if count(V) < minpts then
        continue
    c ← c + 1                          ▷ Start a new cluster
    for j ← [1, n] do
        if V[j] = true ∧ cluster[j] ≠ 0 then
            cluster[j] ← c
            s.push(j)
    while s ≠ ∅ do
        k ← s.pop()
        V ← getNeighbors(k, P, minpts, radius)
        if count(V) < minpts then
            continue
        for j ← [1, n] do
            if V[j] = true ∧ cluster[j] ≠ 0 then
                cluster[j] ← c
                s.push(j)
```

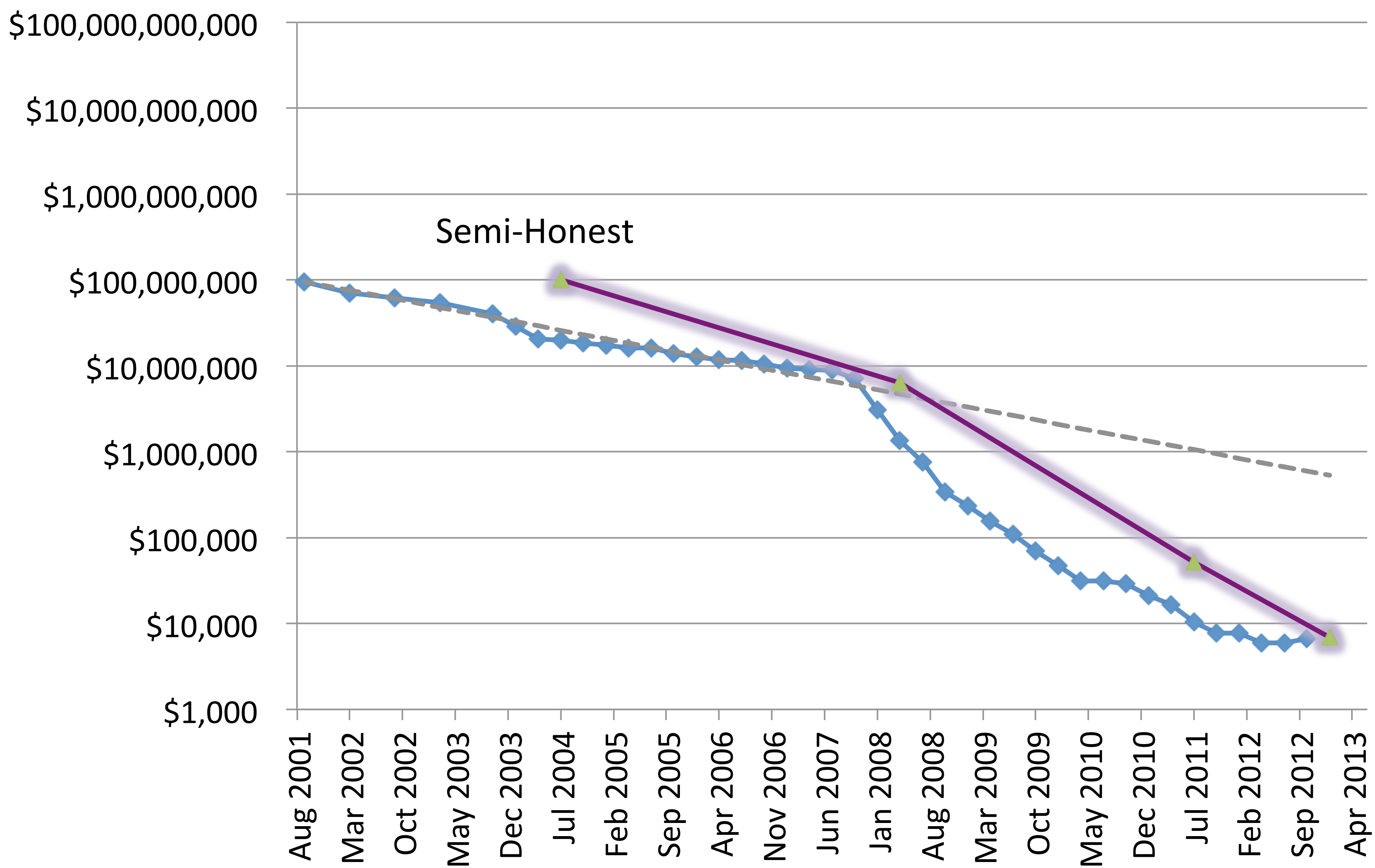
Conditional Push! (s.push(j))

Array update! (cluster[j] ← c)

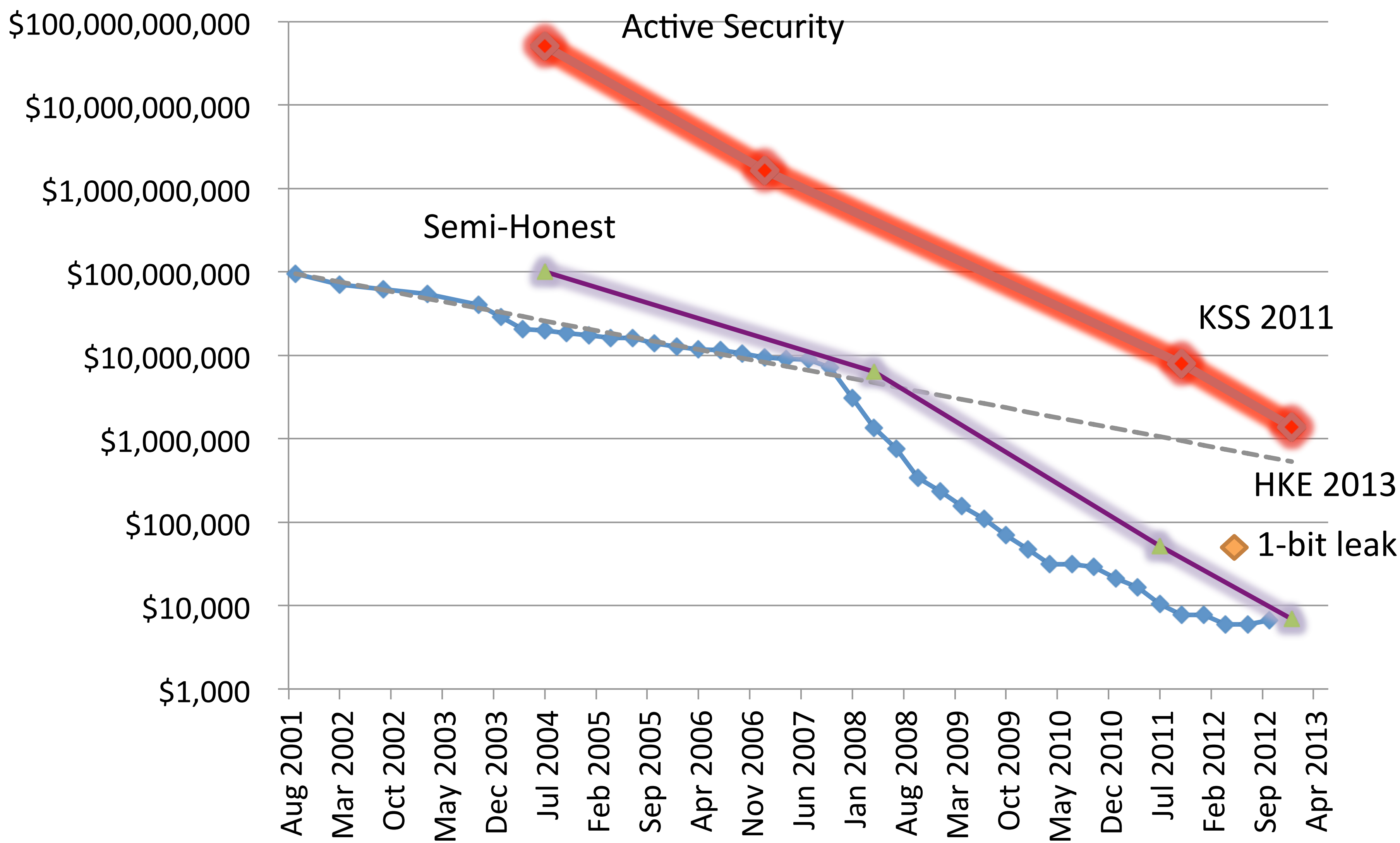

Execution Time (seconds)
40000
35000
30000
25000
20000
15000
10000
5000
0
60
120
240
480
Data Size
9.7 hours
Normal Data Structures
Optimized Structures
55 minutes


$100,000,000
$10,000,000
$1,000,000
$100,000
$10,000
$1,000
Moore's Law prediction
(halve every 18 months)
Cost to sequence human genome
Aug 2001
Mar 2002
Oct 2002
May 2003
Dec 2003
Jul 2004
Feb 2005
Sep 2005
Apr 2006
Nov 2006
Jun 2007
Jan 2008
Aug 2008
Mar 2009
Oct 2009
May 2010
Dec 2010
Jul 2011
Feb 2012
Sep 2012
Apr 2013


FairPlay (2004) [10k*10k alignment]
Free XOR
HEKM
Schneider & Zhoner 2013
$100,000,000
$10,000,000
$1,000,000
$100,000
$10,000
$1,000
Aug 2001
Mar 2002
Oct 2002
May 2003
Dec 2003
Jul 2004
Feb 2005
Sep 2005
Apr 2006
Nov 2006
Jun 2007
Jan 2008
Aug 2008
Mar 2009
Oct 2009
May 2010
Dec 2010
Jul 2011
Feb 2012
Sep 2012
Apr 2013



$100,000,000,000
$10,000,000,000
$1,000,000,000
$100,000,000
$10,000,000
$1,000,000
$100,000
$10,000
$1,000
Semi-Honest
Aug 2001
Mar 2002
Oct 2002
May 2003
Dec 2003
Jul 2004
Feb 2005
Sep 2005
Apr 2006
Nov 2006
Jun 2007
Jan 2008
Aug 2008
Mar 2009
Oct 2009
May 2010
Dec 2010
Jul 2011
Feb 2012
Sep 2012
Apr 2013



Active Security
Semi-Honest
KSS 2011
HKE 2013
1-bit leak
$100,000,000,000
$10,000,000,000
$1,000,000,000
$100,000,000
$10,000,000
$1,000,000
$100,000
$10,000
$1,000
Aug 2001
Mar 2002
Oct 2002
May 2003
Dec 2003
Jul 2004
Feb 2005
Sep 2005
Apr 2006
Nov 2006
Jun 2007
Jan 2008
Aug 2008
Mar 2009
Oct 2009
May 2010
Dec 2010
Jul 2011
Feb 2012
Sep 2012
Apr 2013

mightbeevil.com

Questions?