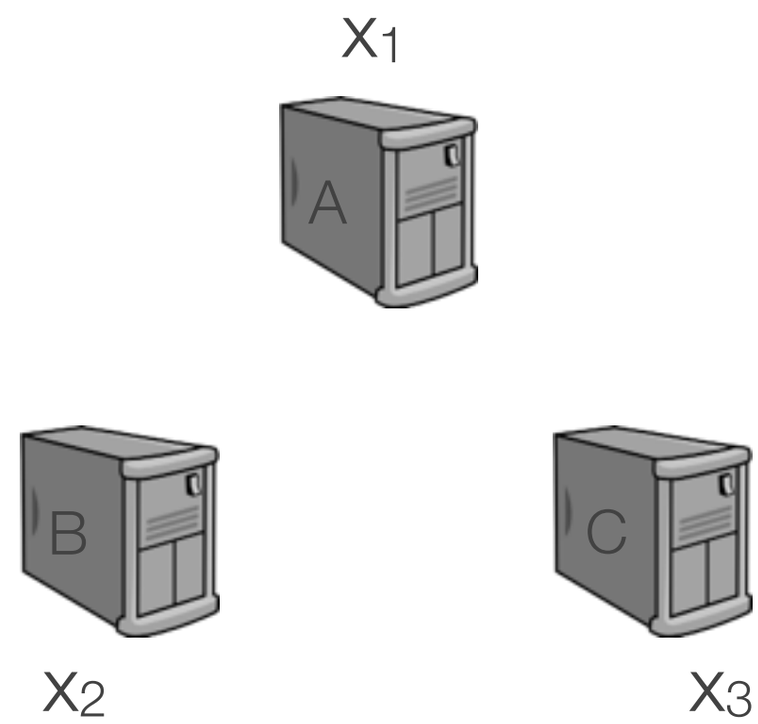
galois

(Some) Practical Optimizations for Secret Sharing MPC

Dave Archer, Tom DuBuisson, John Launchbury, Eric Mertens
Galois, Inc.

Context: Linear Secret Sharing

$x = (x_1 + x_2 + x_3) \bmod 2^n$



- Each x_i is locally random
- Together, globally meaningful
- Linear, so interpolation simplifies to addition for recovering values

- HBC adversary (for now)
- Secure channels
- Gain security within HBC by:
  - Different architectures
  - Different administrators
  - Different locations
- Or implement more robust model

LSS Addition (or XOR)

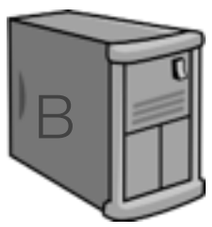
x + y

$x = (x_1 + x_2 + x_3) \bmod 2^n$

$y = (y_1 + y_2 + y_3) \bmod 2^n$

$x_1 + y_1$

$x+y = (x_1 + y_1) \bmod 2^n + (x_1 + y_1) \bmod 2^n + (x_1 + y_1) \bmod 2^n$



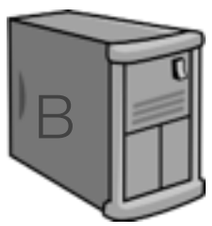
$x_2 + y_2$

$x_3 + y_3$

LSS Multiplication (or AND)

x * y

x_1, y_1



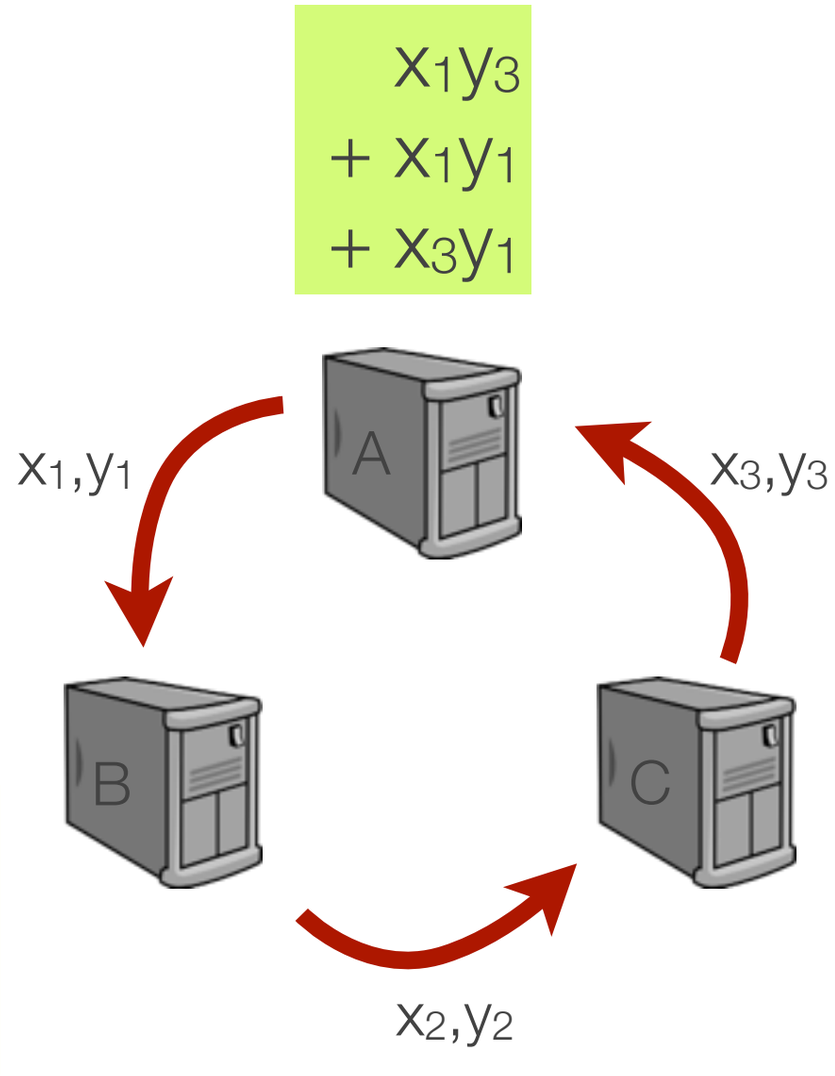
x_2, y_2

x_3, y_3

*	y	y	y
x	x	x	x
x	x	x	x
x	x	x	x
x	x	x	x

Multiplication (or AND)

x * y



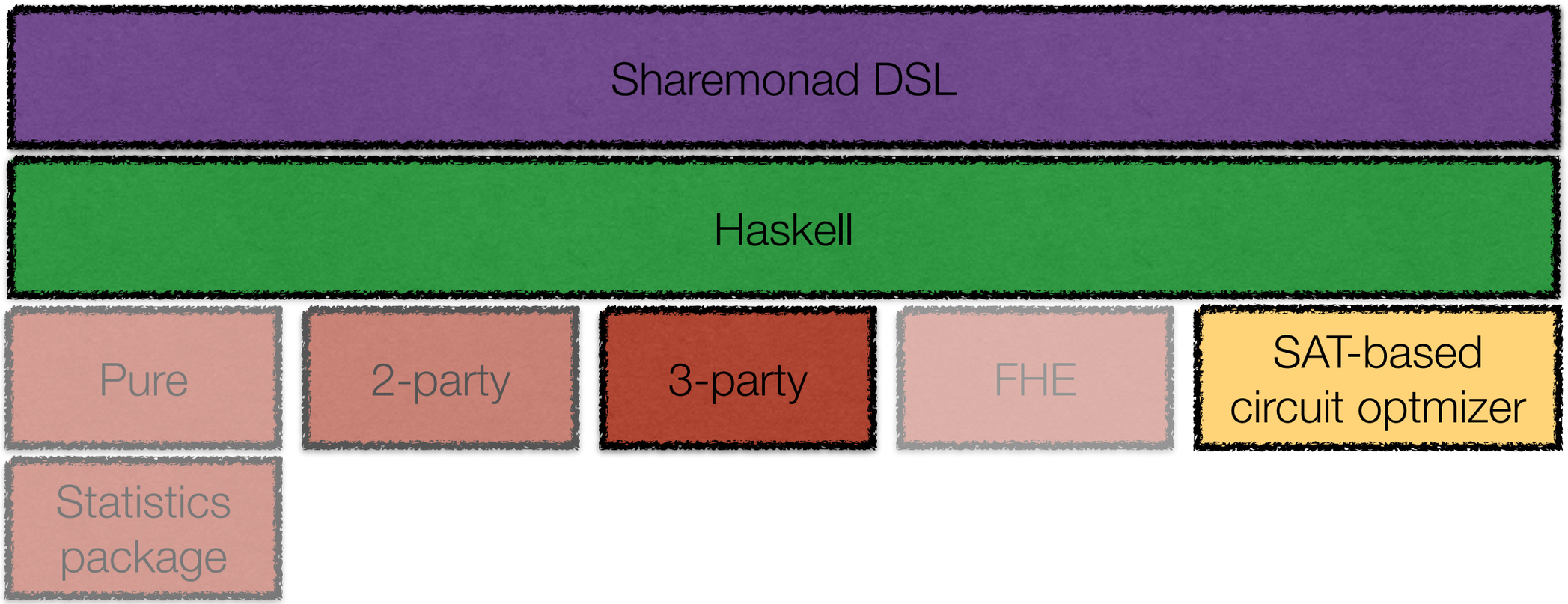
*	y	y	y
x	x	x	x
x	x	x	x
x	x	x	x
x	x	x	x

New entropy during re-share operations prevents A, B, C from learning secrets

Promising or Practical?

- AES-128 @ 3ms per block — 10^5 too slow
- Corollary of Felten: adding security not compelling
- No “killer apps” publicly identified (yet)
- Our hypothesis: practical applications with compelling benefit
  - Streaming-rate, on-demand cloud privacy
  - Protecting intellectual property AND privacy in e-mail

Sharemonad MPC Machine



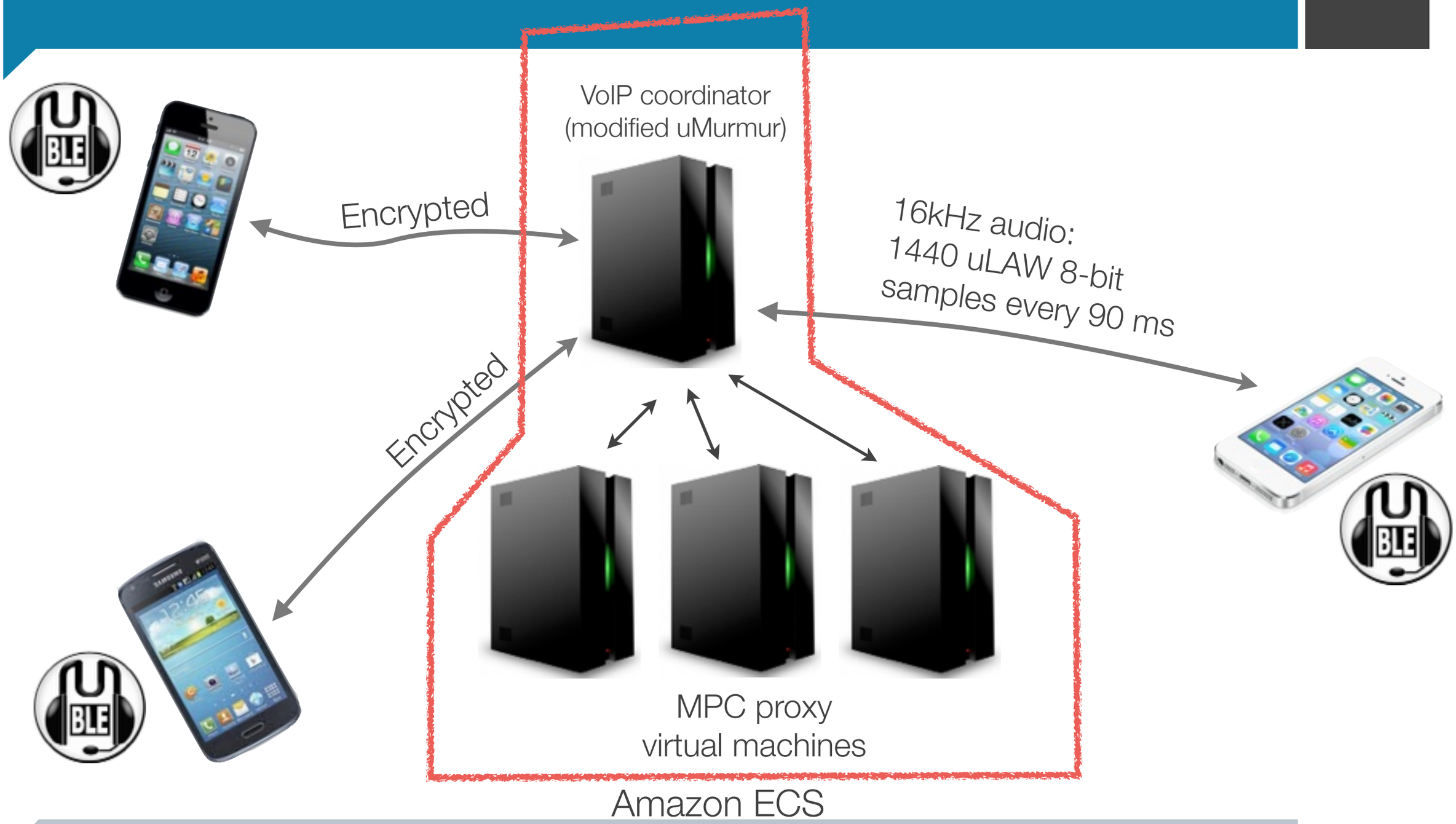
Sharemonad eDSL

- Secret-sharing computation language
- Embedded in Haskell for strong FP benefits, or…
- Ad hoc instruction set architecture
- A Sharemonad program is
  - Generated from a code representation
  - An AST
  - Interpreted as a circuit by a target back-end
- Optimize where? Program level AND circuit level

Sharemonad Primitives

- Data types
  - constant value, including unit and zero
  - GF 2 (bits)
  - GF 2^{32} (integers)
  - Lists of the above types
- Math-like operations on shares
  - multiplication
  - addition, subtraction
  - negation
  - division
- Operations on public values
  - arithmetic mul by public value
  - right bit shift by public value
  - table lookup in public table
- Conversions
  - bit-representation <—> numeric representation
  - list of bits and <—> numeric representation
- Cross-product of bit-wise representation
- Comparisons (eq/lt/lteq/gt/gteq/zero/neg)

On-demand, streaming cloud privacy
Secure VoIP



2-Voice Example

16-bit fixed-point (PCM)

PCM2uLAW

8-bit logarithmic (uLAW) representation

uLAW2PCM

$$F^{-1}(y) = \mathrm{sgn}(y)(1/\mu)((1+\mu)^{|y|} - 1) \qquad -1 \le y \le 1$$

uLAW2PCM

Add and clip

PCM2uLAW

$$F(x) = \mathrm{sgn}(x)\frac{\ln(1+\mu|x|)}{\ln(1+\mu)} \qquad -1 \le x \le 1$$

uLAW2PCM

16-bit fixed-point (PCM)

PCM2uLAW

8-bit logarithmic (uLAW) representation

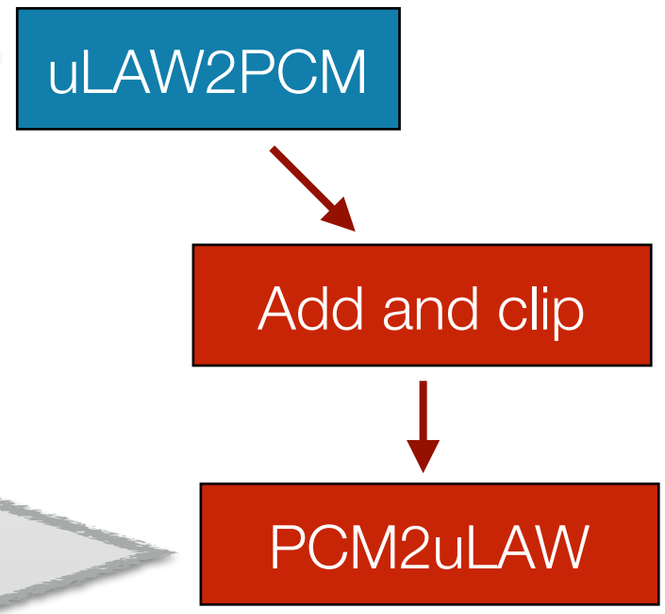
uLAW2PCM

First Attempt: Algorithmic

Local **Global**

```
ulawbyte = ~ulawbyte;
sign = (ulawbyte & 0x80);
short tempsign = sign | sign >> 1;
tempsign |= tempsign >> 2;
tempsign |= tempsign >> 4;
tempsign = tempsign | tempsign << 8;
short adder = tempsign & 0x0001;
mantissa = ulawbyte & 0x0F;
exponent = ulawbyte & 0x70;
exponent = exponent >> 4;
short expPlus3 = exponent + 3;
short tempMantissa = mantissa << expPlus3;
sample = exp_lut[exponent]; // oblivious table lookup
sample = sample + tempMantissa;
sample = sample ^ tempsign;
sample = sample + adder;
```

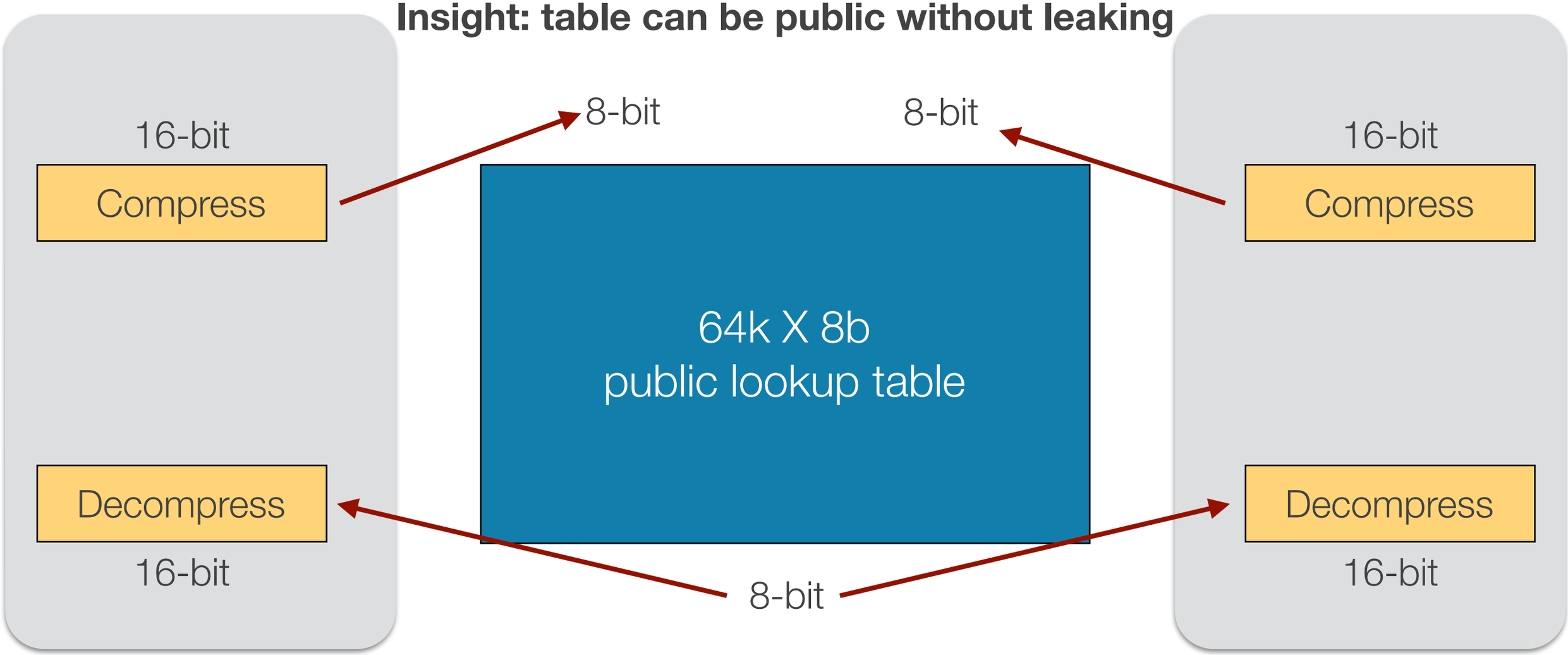
Similar,
but inverse



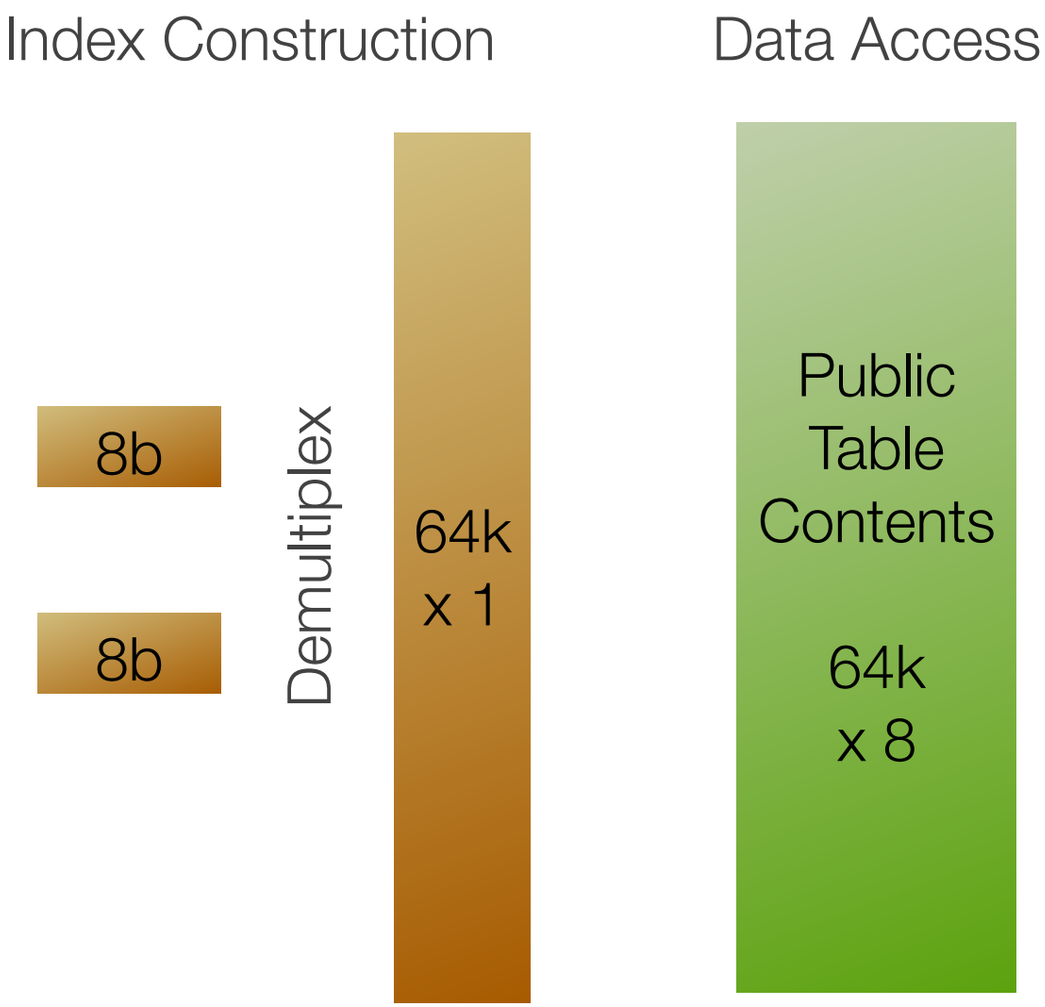
1440 samples per packet
4 virtual voice processors
12 seconds (!) vs. 90ms goal

Insight and Second Attempt

Insight: narrow inputs, narrow output
Insight: table can be public without leaking

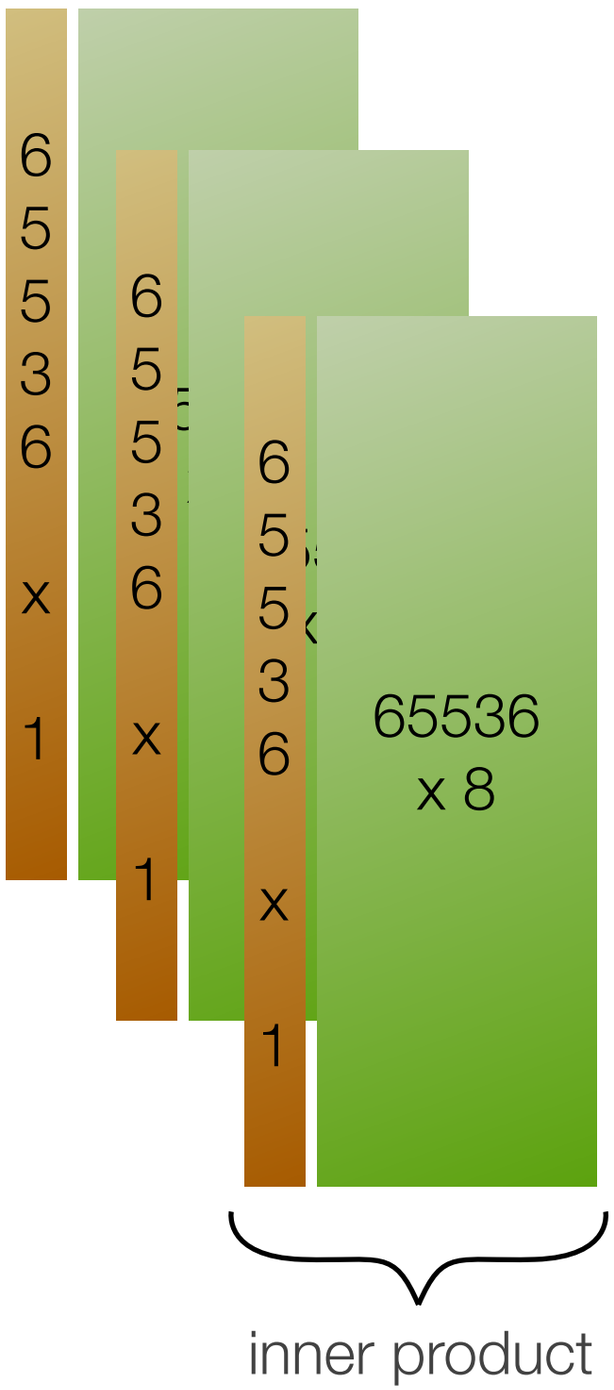


Two Parts to Table Lookups



Approach:
Public table shared by all proxies
Each lookup must access entire table
Invariant: 3 shares add to correct intermediate result

Data Access



- Demuxed index *d* is shared, table *t* is public
- n-1 index bits have even parity across shares
- 1 index bit has odd parity

$$d = 0000000010000000$$
$$d_A = 1011001011101011$$
$$d_B = 0011010011001101$$
$$d_C = 1000011010100110$$

- Inner product
  - Whole table structure is traversed locally
  - All table rows except valid one will cancel

$$d = d_A \hat{} d_B \hat{} d_C$$

implies

$$\text{inner } d\ t = \text{inner } d_A\ t \hat{} \text{inner } d_B\ t \hat{} \text{inner } d_C\ t$$

Index Construction

Divide and conquer:
demux(xs ++ ys) =
demux xs # demux ys

.
64kb
.
256b
.

demux (1001) = 0000 0000 0100 0000

demux(10) # demux(01)

0010 0100

demux(1) # demux(0) demux(0) # demux(1)

01 10 10 01

demux(1) demux(0) demux(0) demux(1)

Bits shared = $O(2^{n/2})$

Index size 2^n

$\log_2(n)$ comms

this step is free…why?

Index Construction Bandwidth

Index size: 2^{16}

n = 16

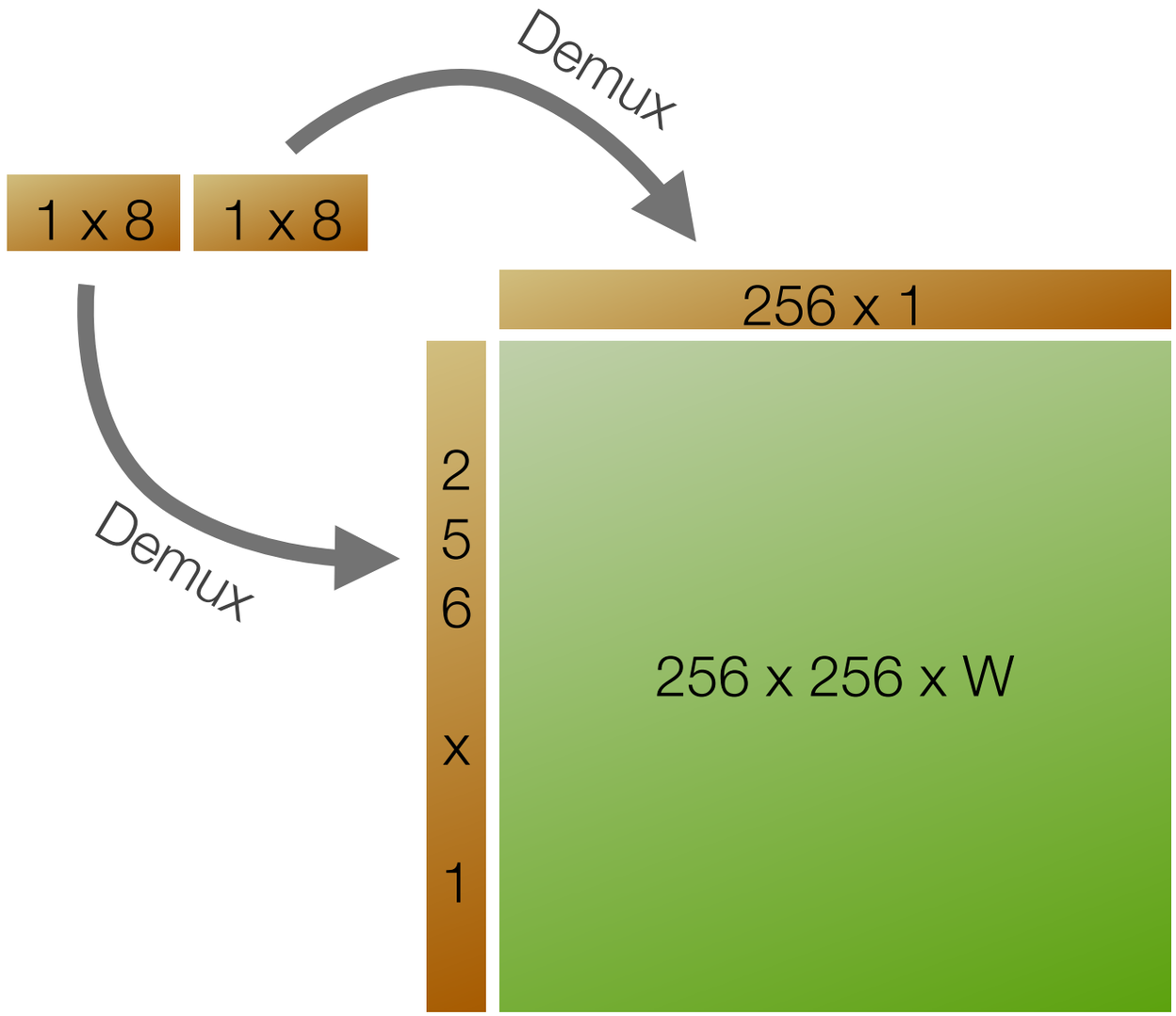
$\log_2(n)$ communication rounds: 4

$O(2^{n/2})$ total bits communicated per server:
256 + 256
16 + 16
4 + 4
2 + 2
(actual number: 700b)

700b x 3 servers = 2kb
2kb x 1440 samples x 1/**25msec** x 4 voice streams processed = **485Mb/s, Oh My!**

Insight and 3rd Attempt

Table is currently 64k entries
Insight: Instead of 64kb index, re-construct table as 256 rows of 256 entries



- Restructure table to 256x256 from 64k x 1
- Share unexpanded final expressions (16b)
- 2-dimensional final lookup against table
  - Cost: 2x as many XORs to compute 2 inner products

Index Bandwidth Again

$O(2^{n/2})$ total bits communicated per server:

16 + 16

4 + 4

2 + 2

~ 32b per server

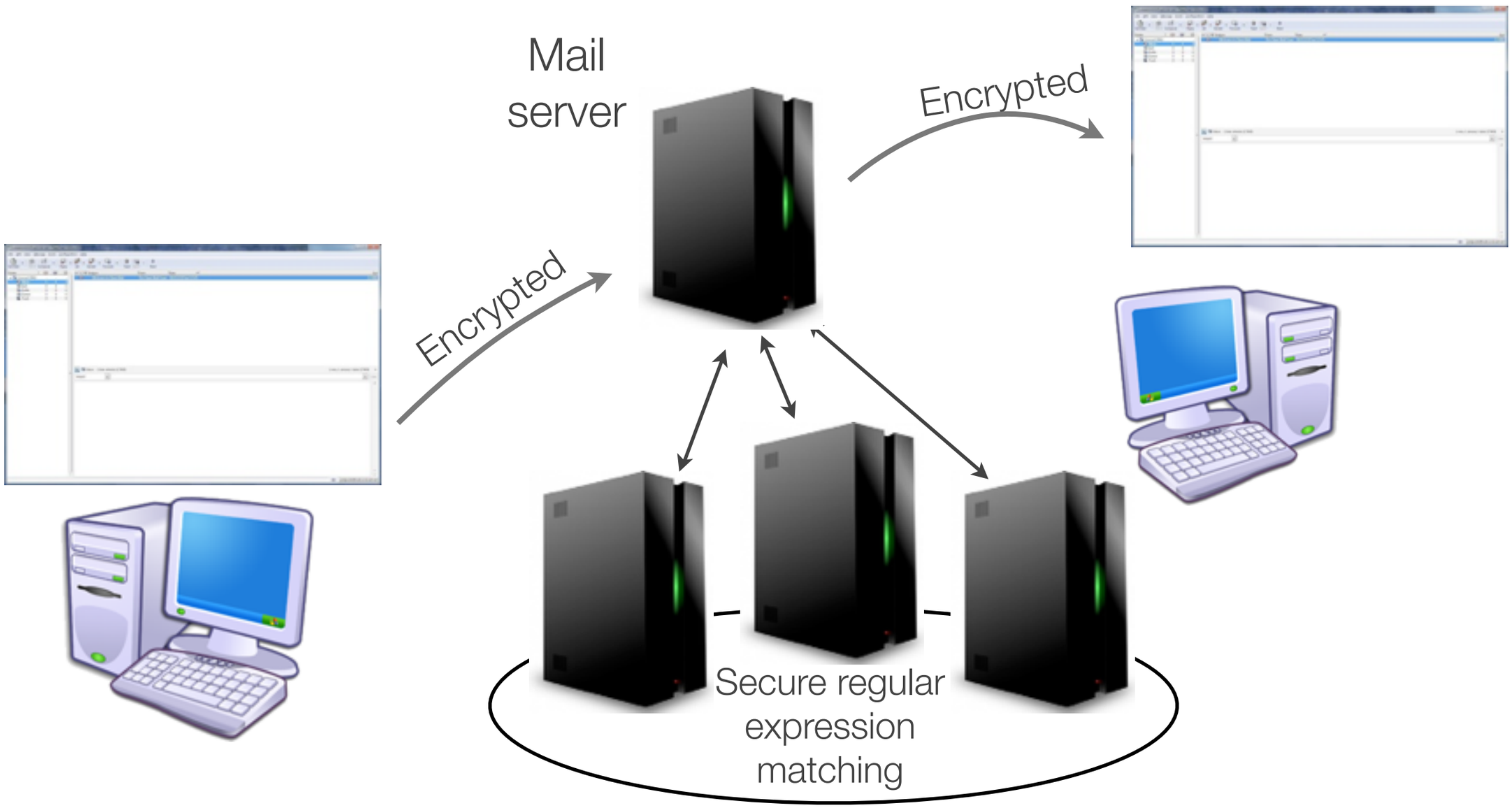
32b x 3 servers = 100b

100b x 1440 samples x 1/25msec x 4 voice streams processed = **2.3 Mb/s**

VoIP Outcome

- 4 voice “accumulators” at 16kB/s sampling rate
- 25ms of processing per incoming 1440-sample block
- 65ms left in for network delay to and from clients
- Amazon ECS VMs for MPC, Apple iPhones for clients

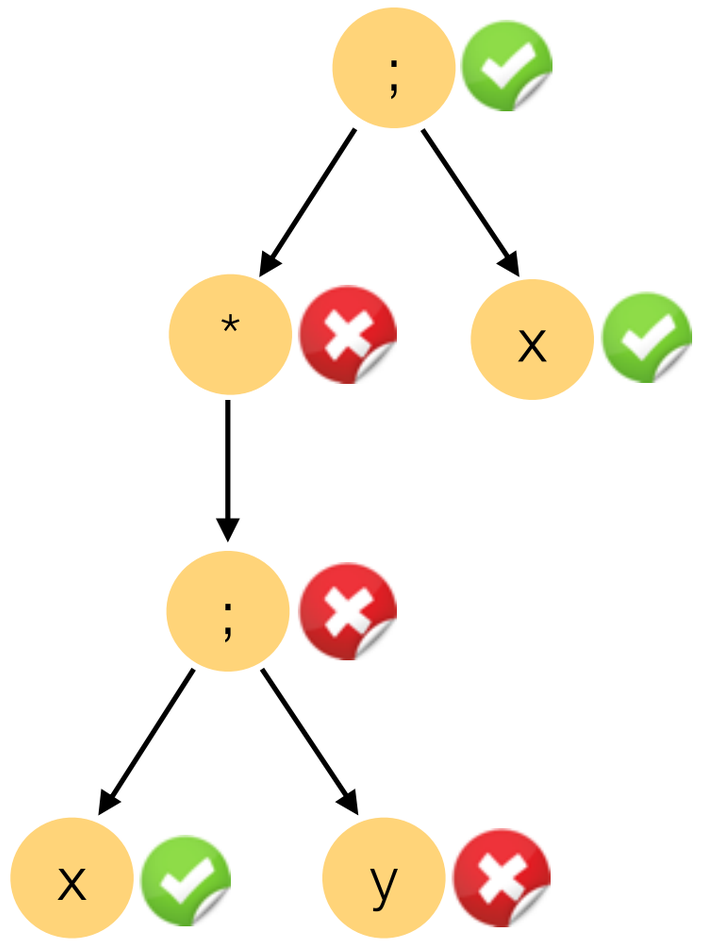
Intellectual Property AND Privacy: eMail border guard



Regular Expression Computation

RE = (x;y)*;x Input = “x”

- Assume
  - Regular expression is public
  - Email string is private (randomly shared)
- Have to compute over all states
  - Non-deterministic finite automaton
- Sequence of “matched” flags form a private (randomly shared) vector
  - 

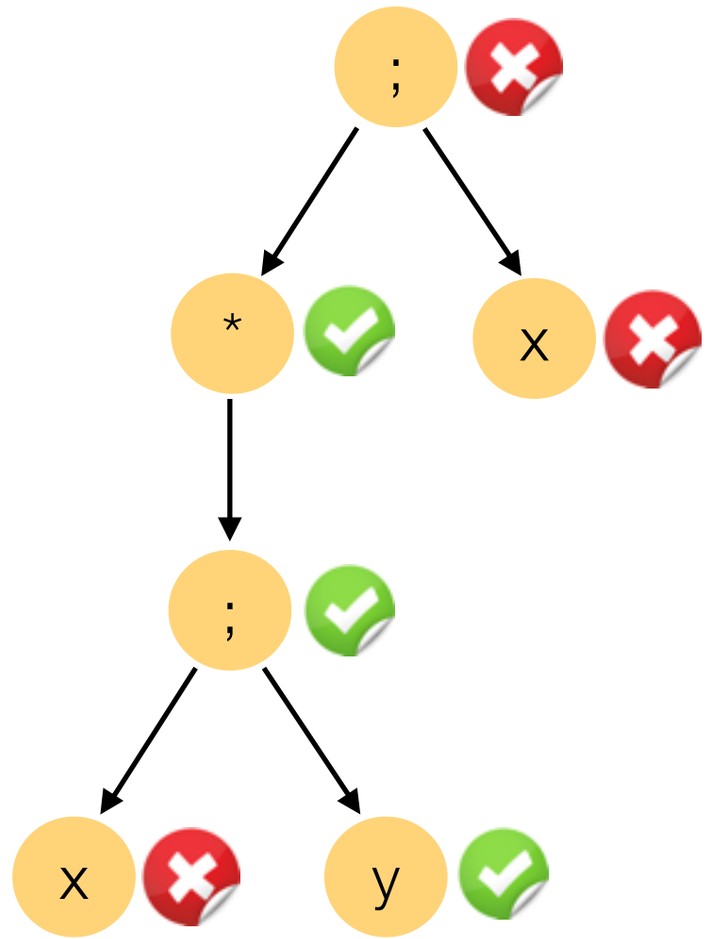


Regular Expression Computation

RE = (x;y)*;x Input = “x”

Next char = ‘y’

- Assume
  - Regular expression is public
  - Email string is private (randomly shared)
- Have to compute over all states
  - Non-deterministic finite automaton
- Sequence of “matched” flags form a private (randomly shared) vector
  - 
- Each new input character causes the match state-vector to be updated
  - 



Circuit Optimization

Goal 1: Eliminate communications (ANDs)

SAT-based circuit optmizer

- Replace AND with XOR

  `if (a | b) then (a & b) ===> ~(a^b)`

- Redundant multiplicands

  `if (a => b) then (a & b) ===> a`

- Associativity and commutativity

  `(a & (b & a)) ===> (a & b)`

- Factorization

  `(a & b) ^ (a & c) ===> a & (b^c)`

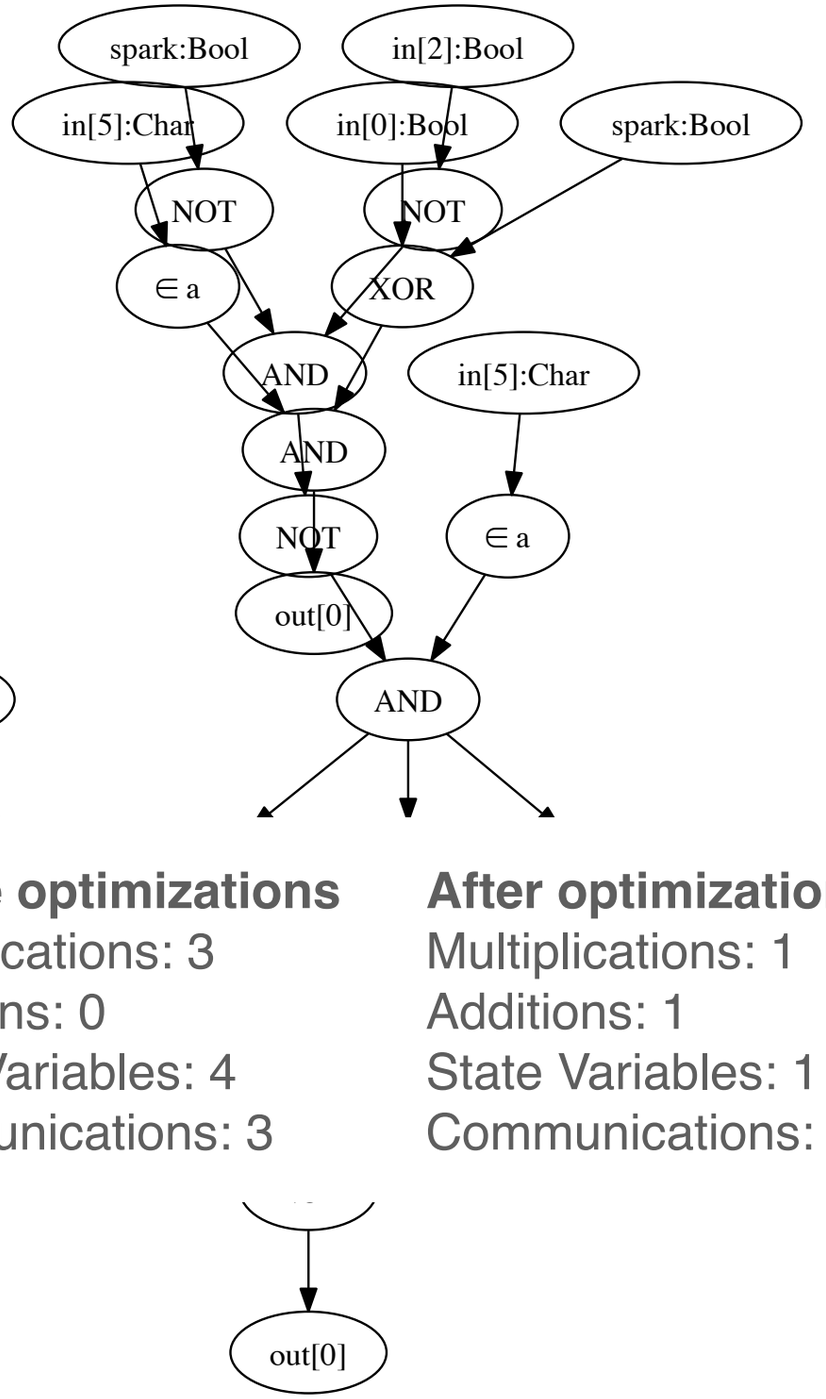
- Idempotence

  `(a & a) ===> a`

  `(a ^ a) ===> false`

- Constant value

  `(x=='a') & (x=='b')  ===> false`

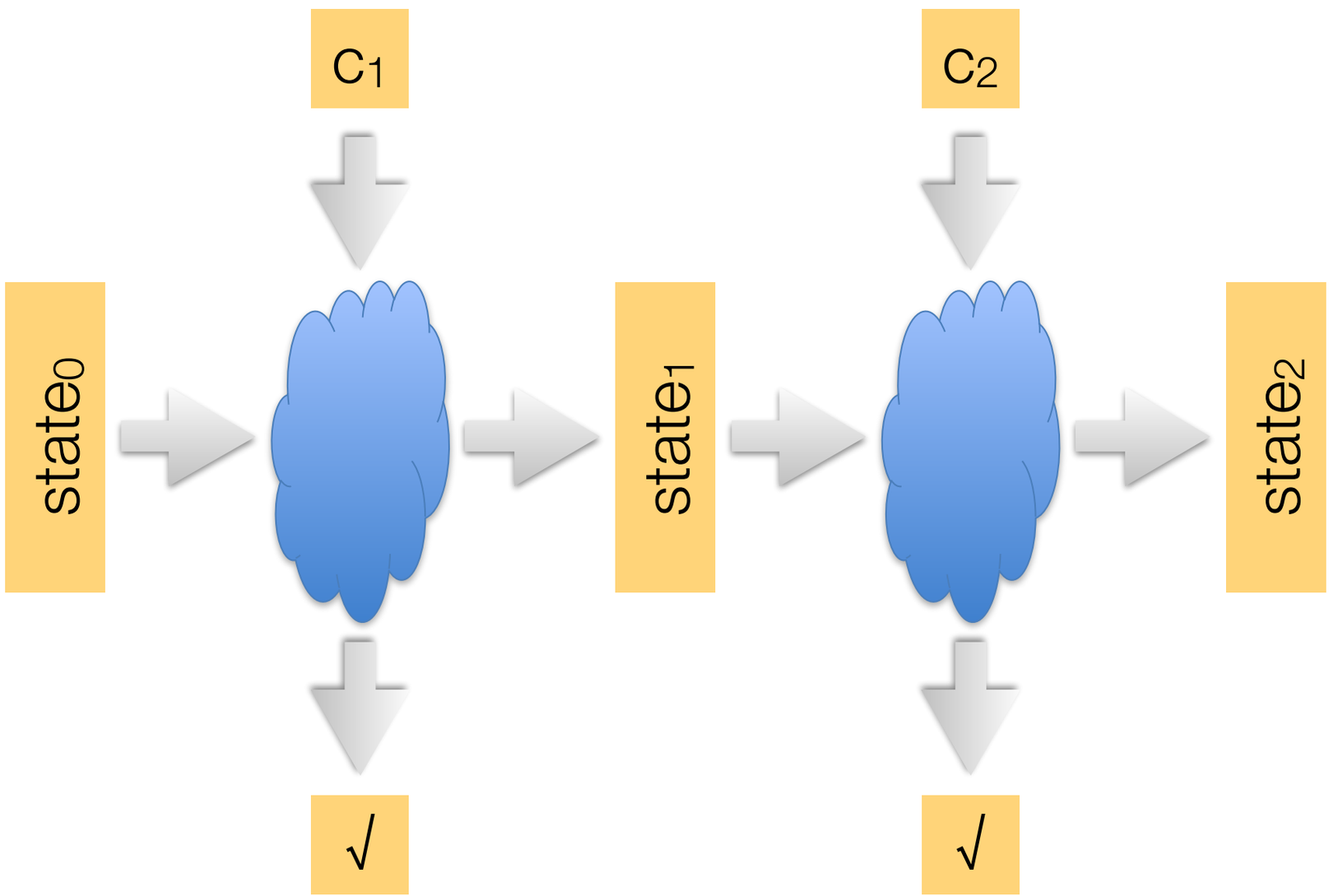


Before optimizations
Multiplications: 3
Additions: 0
State Variables: 4
Communications: 3

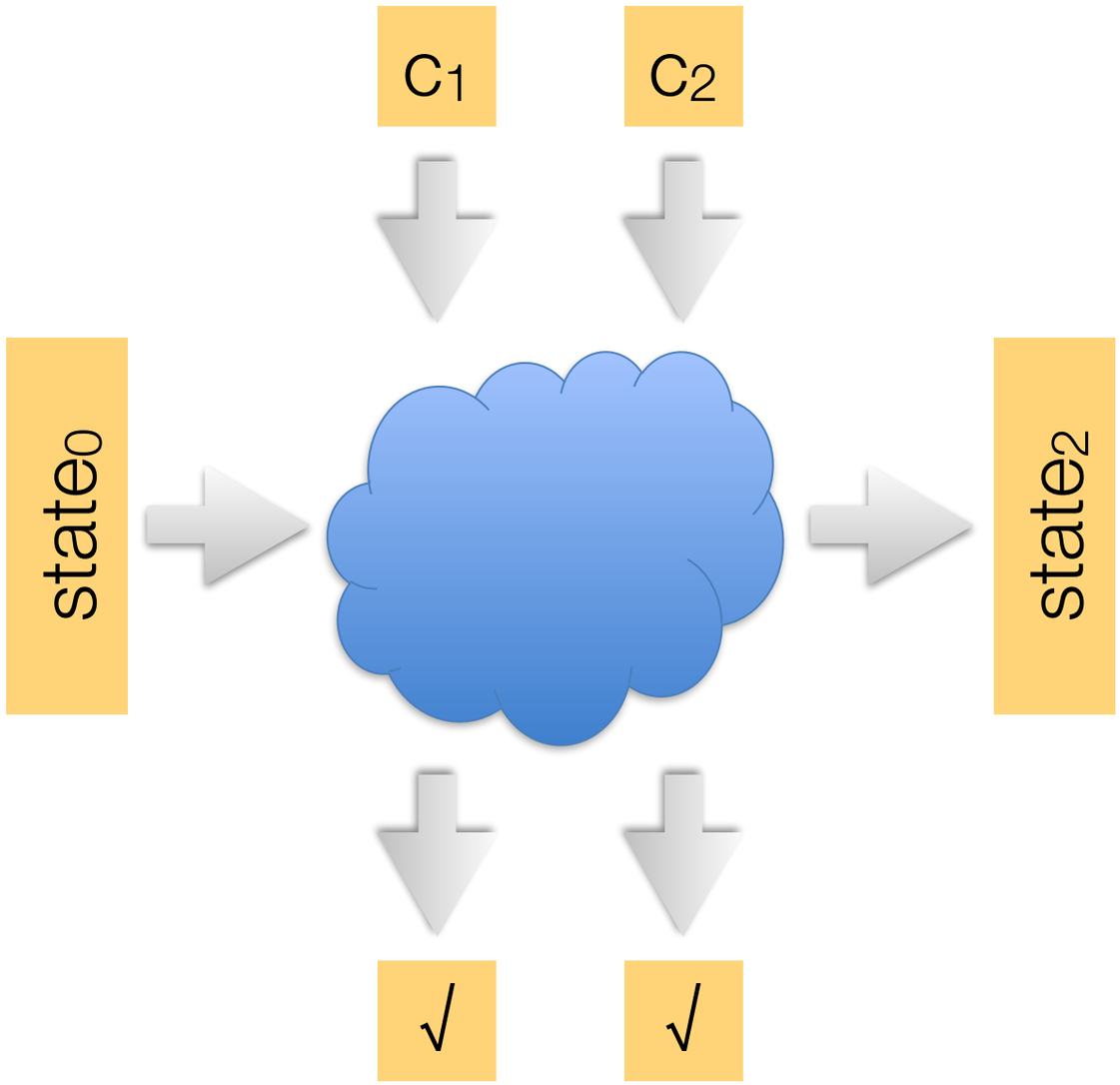
After optimizations
Multiplications: 1
Additions: 1
State Variables: 1
Communications: 1

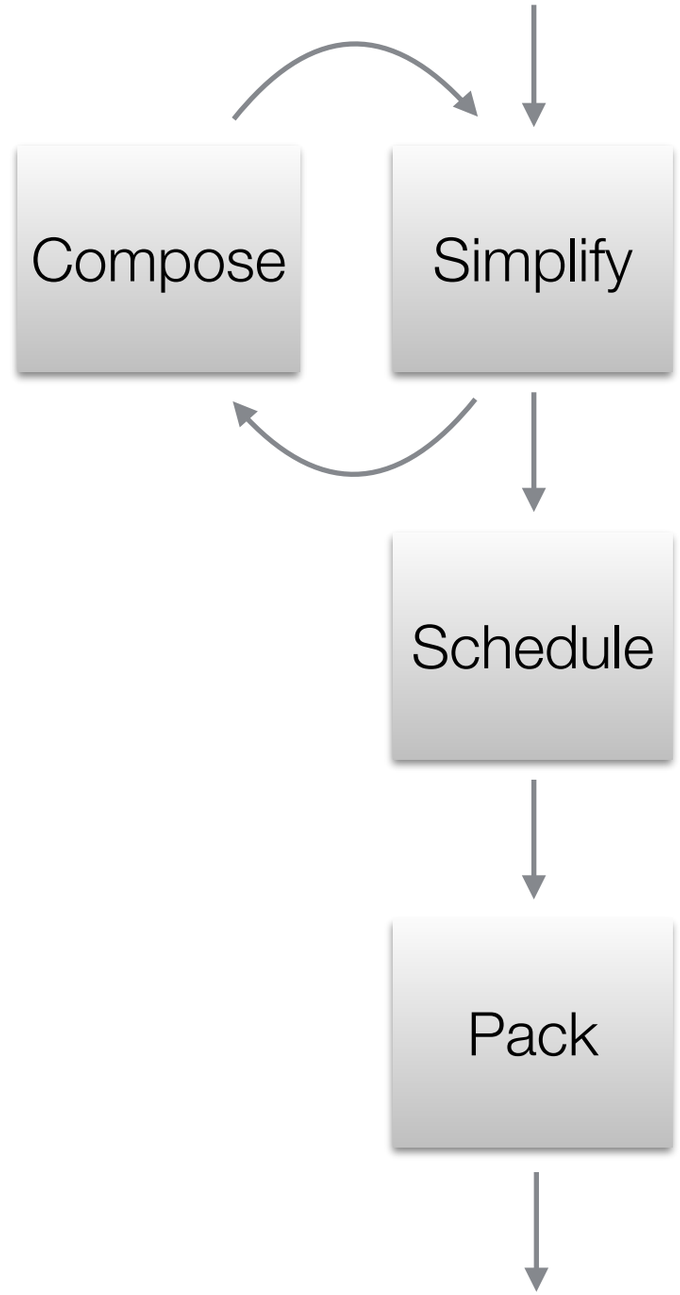
Iterate the Regexp Computation

Goal 2: Expand and re-optimize
Possible analogy: loop unrolling



Simplify the Larger Description





Effect of Circuit Reduction

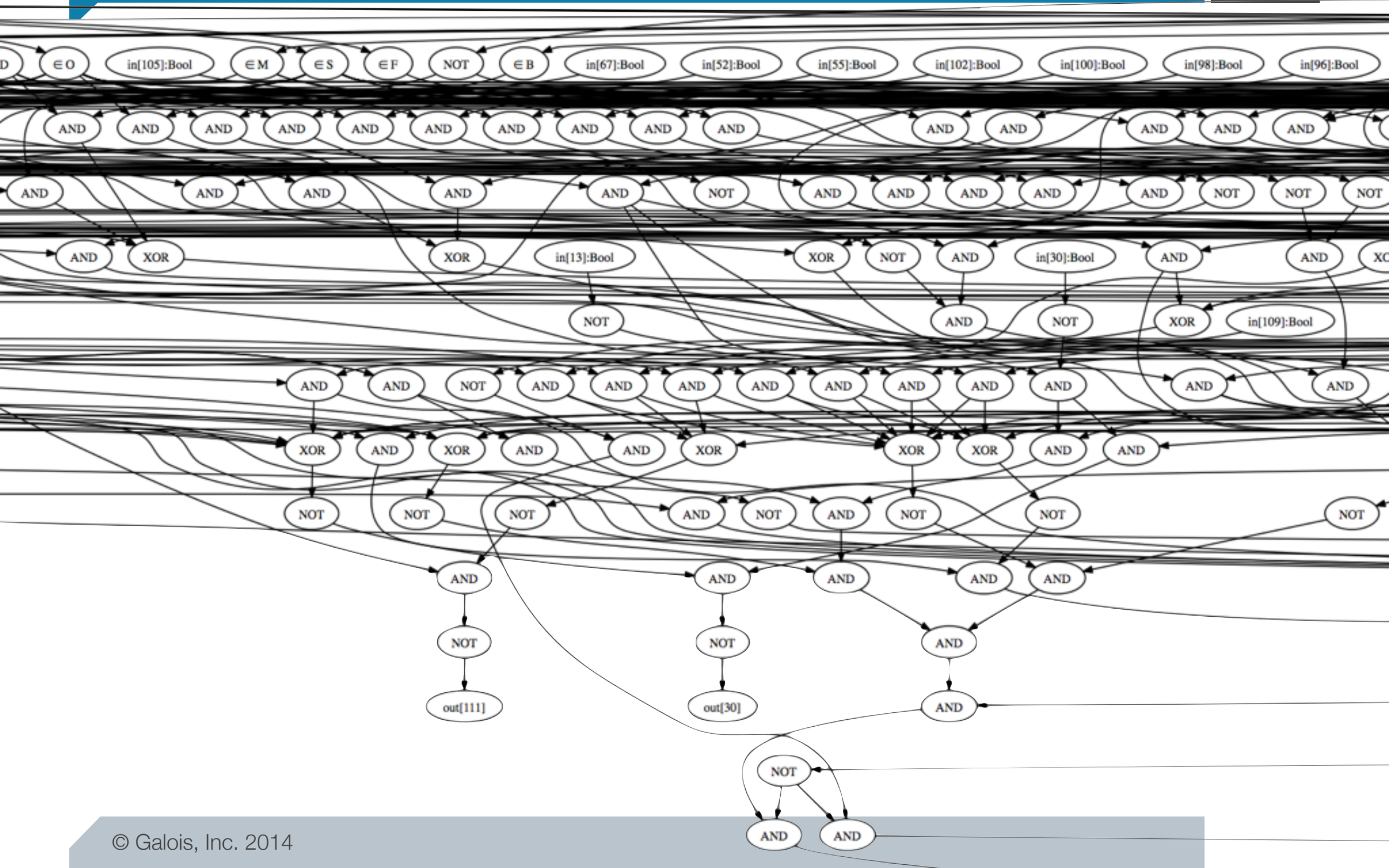
- Compile multiple (1, 2, 4, 8,...) state steps (corpus characters) as a single circuit
- Organize into communication layers; pack bits into words

	unoptimized				optimized			
input	**ands**	**xors**	**state**	**comms**	**ands**	**xors**	**state**	**comms**
1	203	0	358	10	149	15	119	4
2	388	0	358	12	277	27	117	5
4	756	0	358	14	493	53	117	6
8	1492	0	358	19	949	104	117	9
16	2964	0	358	33	**1,950**	212	**117**	**17**

Diminishing Returns

```
.*(((TOP|)SECRET)|TS|S)--(ROCKYBEACH|STINGRAY).*
.*(((TOP|)SECRET)|TS|S)--SI--NO(CON|CONTRACTOR|FOREIGN).*
.*(((TOP|)SECRET)|TS|S|R|RESTRICTED)--(AO|DO|MO|SO|TO)--LIMDIS.*
.*ac*cb.*
```

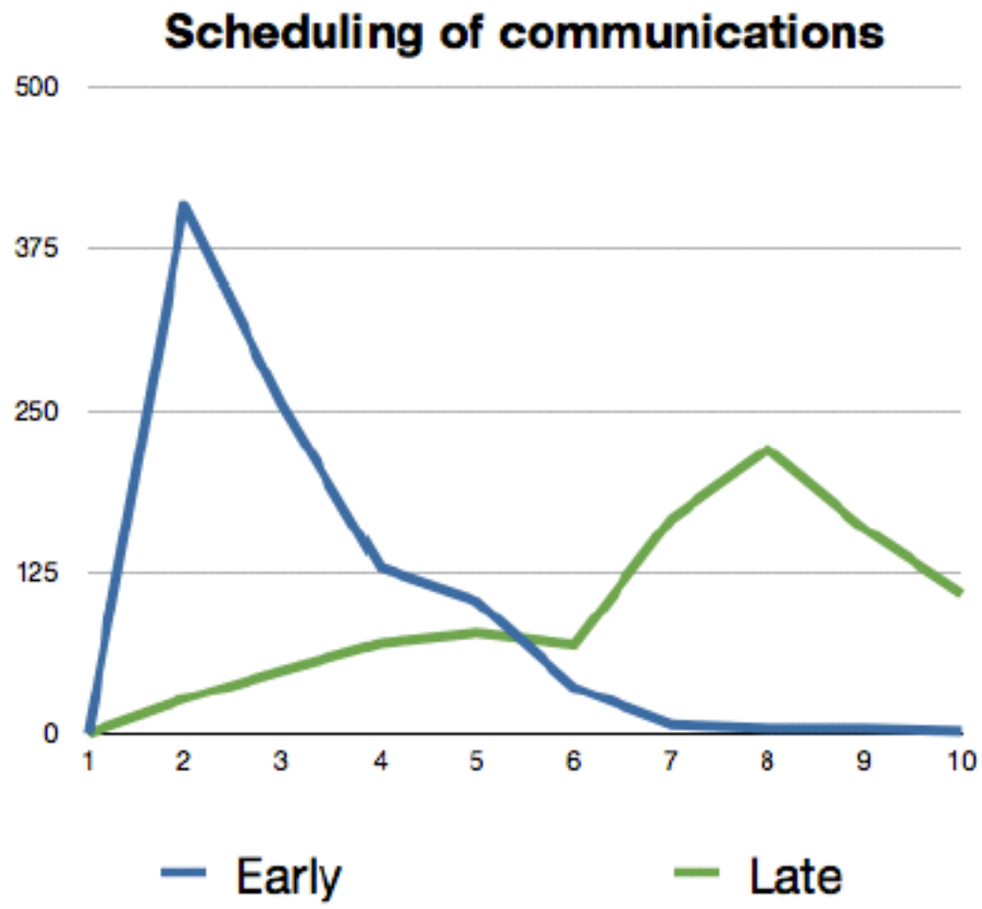
Grouping



Email Border Guard Outcome

- 1-page e-mail
- Classification-marking regexp filter
- ~15s to accept/reject
- Private cloud VMs for MPC, Linux laptops for clients

Resource Scheduling



Conclusions, Future Work

- Optimizations *can* bring interesting apps to practice
- Communication is often the bottleneck (for LSS at least)
- But…in many settings, traditional crypto protocols are cheaper and faster

- Futures
  - Additional instructions for our MPC machine
  - Automation for
    - Determination of diminishing returns for circuit optimization
    - Decisions on LUT replacement of instruction sequences
    - Scheduling of communications among share servers
  - Stronger than HBC adversaries

Thank you!