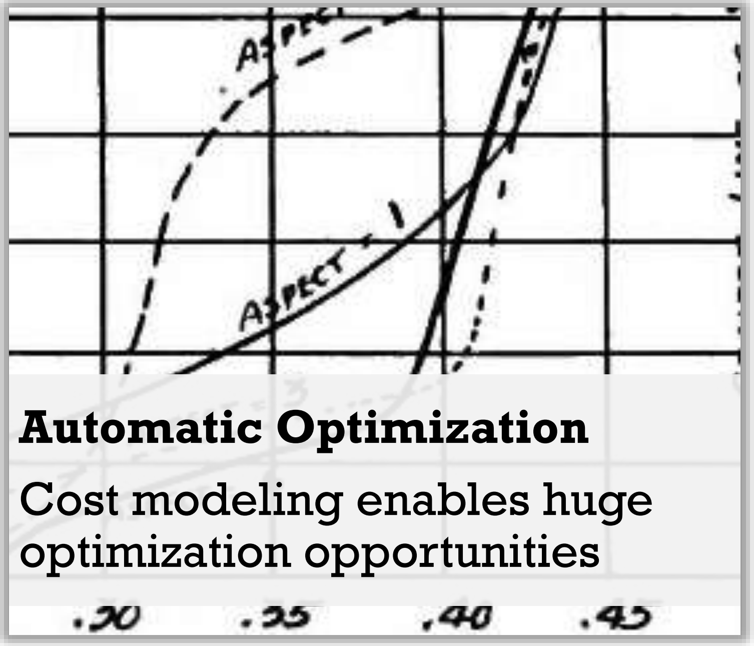
ZØ: An Optimizing Distributing Zero-Knowledge Compiler

Matt Fredrikson
University of Wisconsin

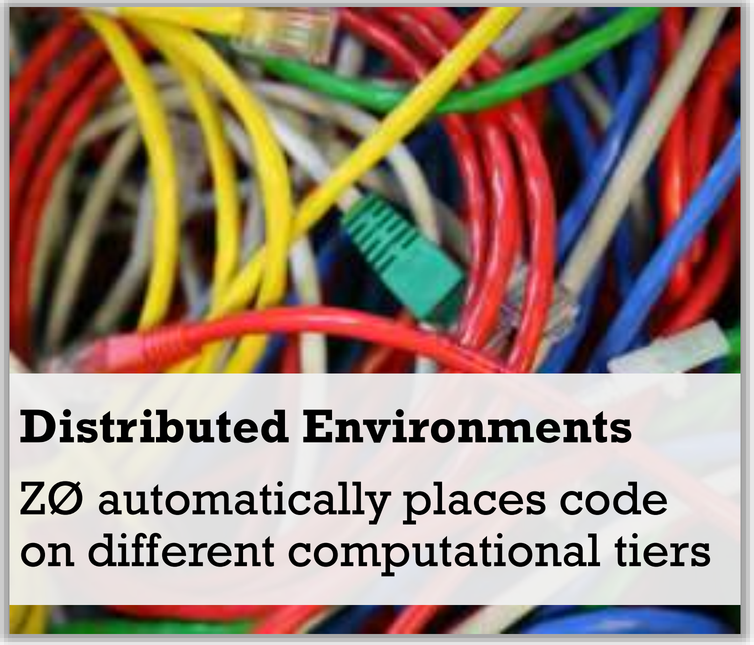
Ben Livshits
Microsoft Research

This talk: at a glance



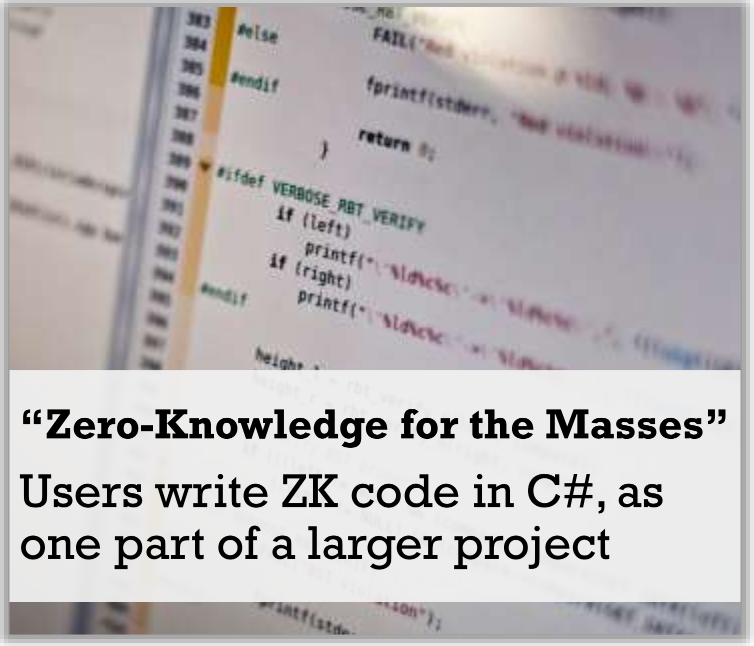
Automatic Optimization

Cost modeling enables huge optimization opportunities



Distributed Environments

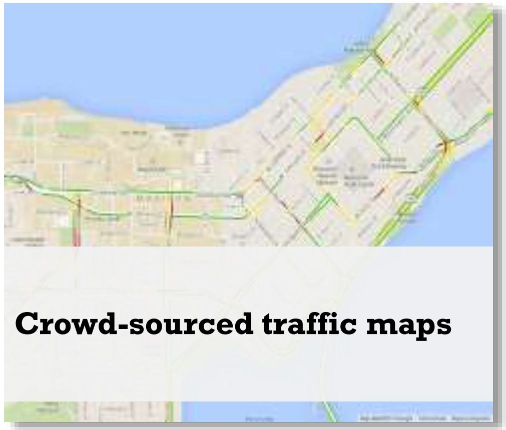
ZØ automatically places code on different computational tiers



"Zero-Knowledge for the Masses"

Users write ZK code in C#, as one part of a larger project

This talk: at a glance

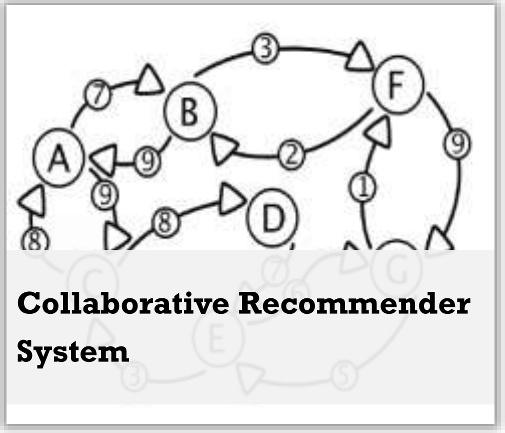


Crowd-sourced traffic maps

Personal Fitness Rewards

Retail Loyalty Card

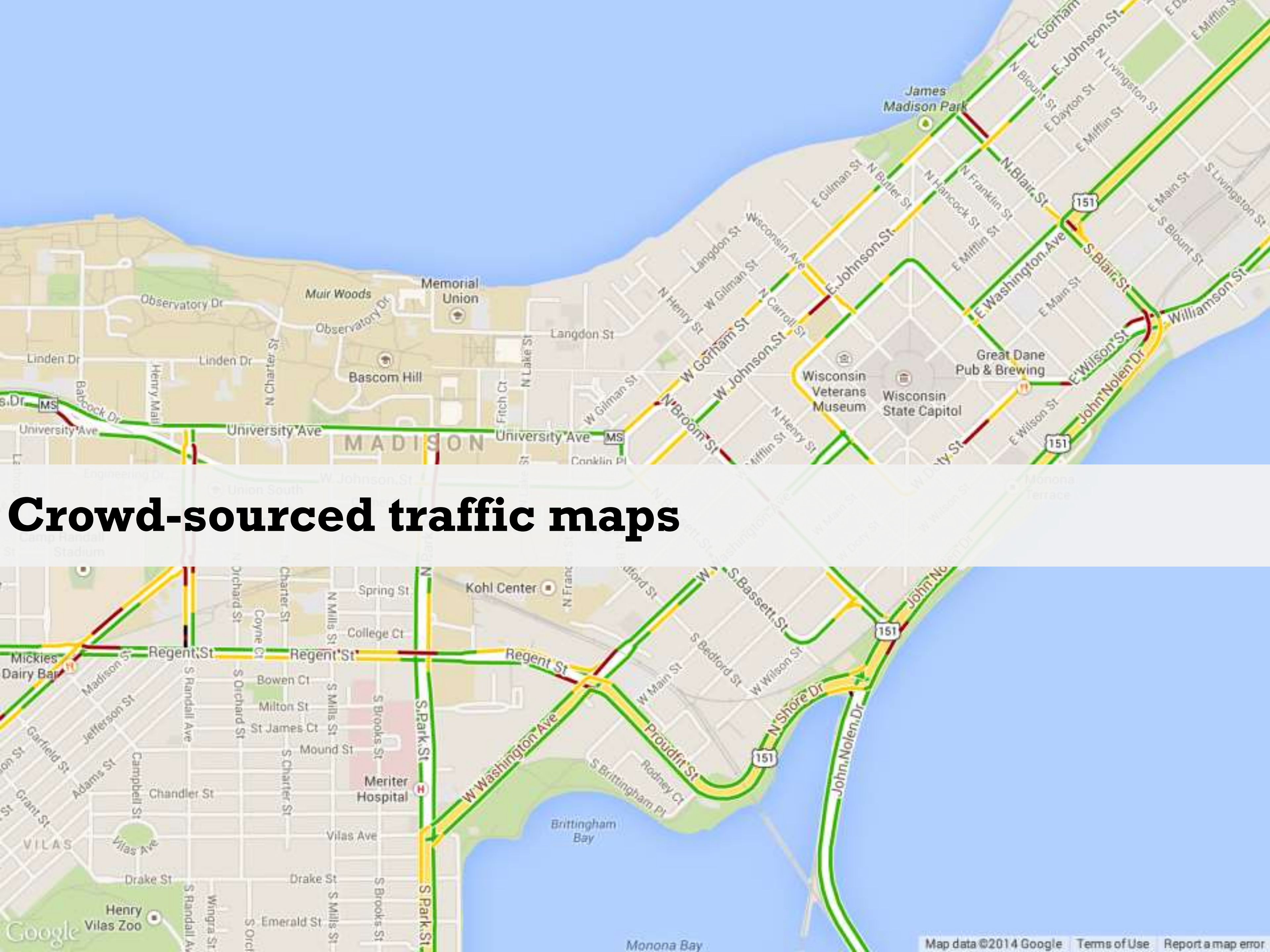
Human Subjects Studies

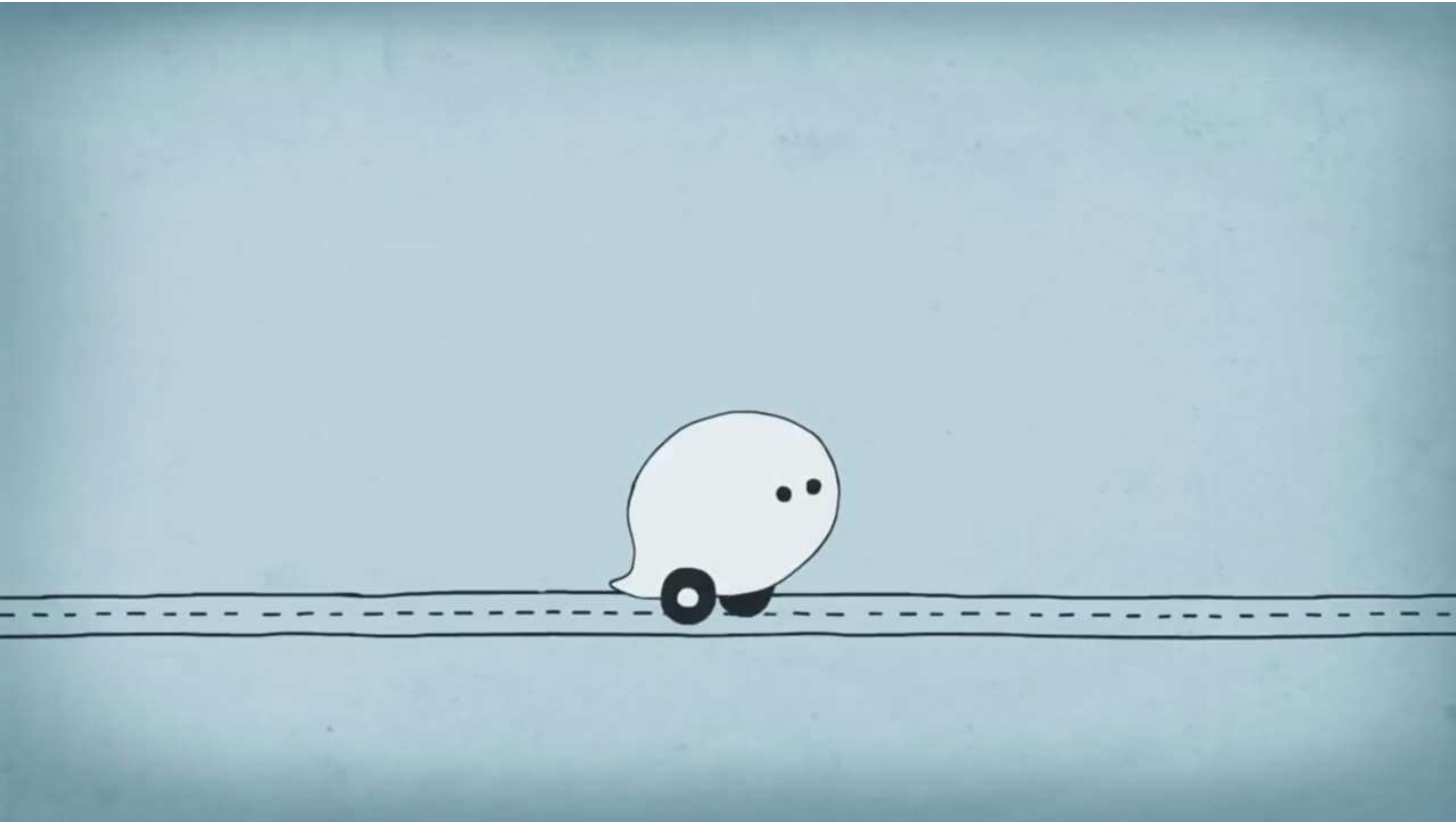


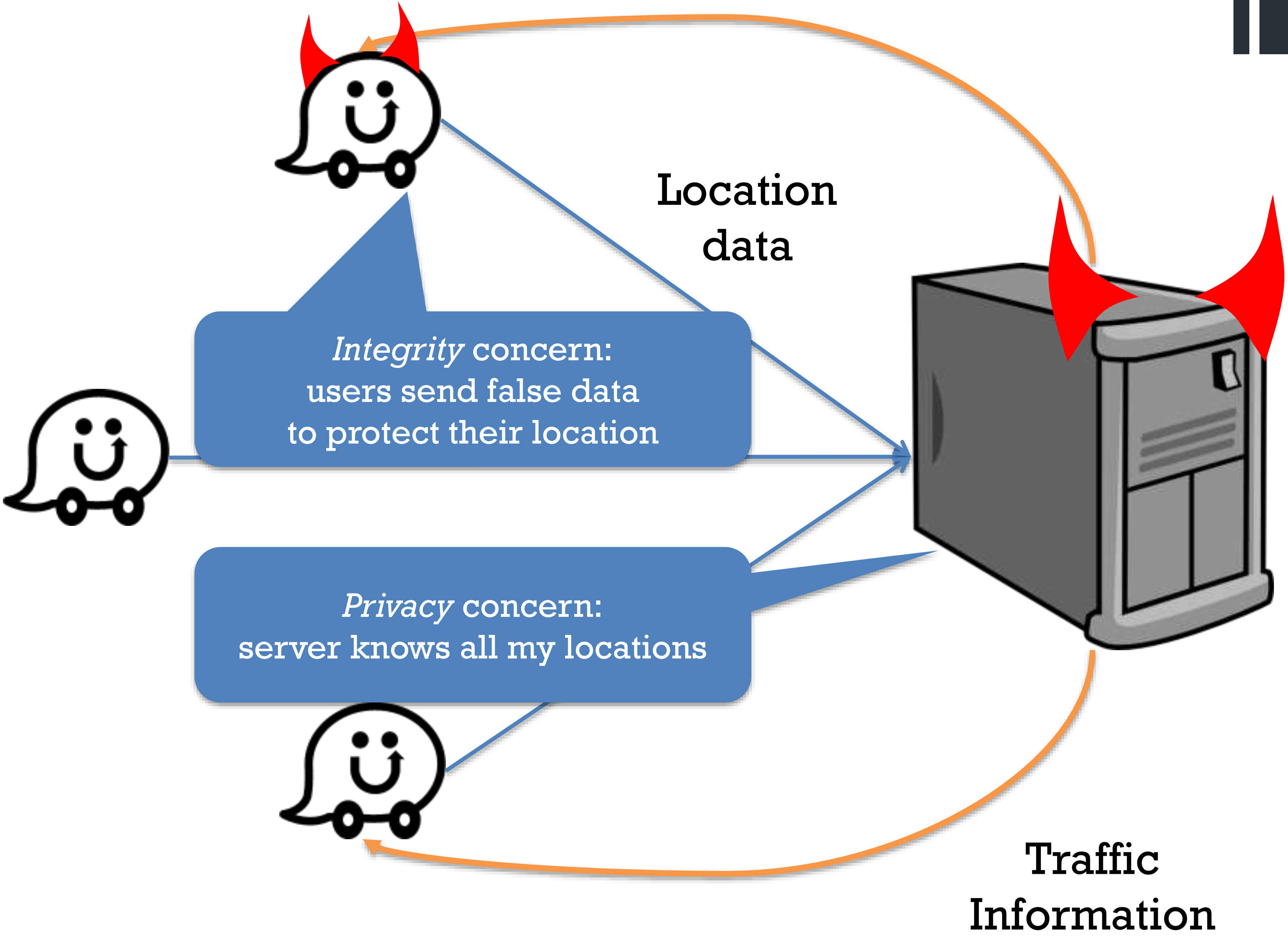
Collaborative Recommender System

Collaborative NIDS

Crowd-sourced traffic maps





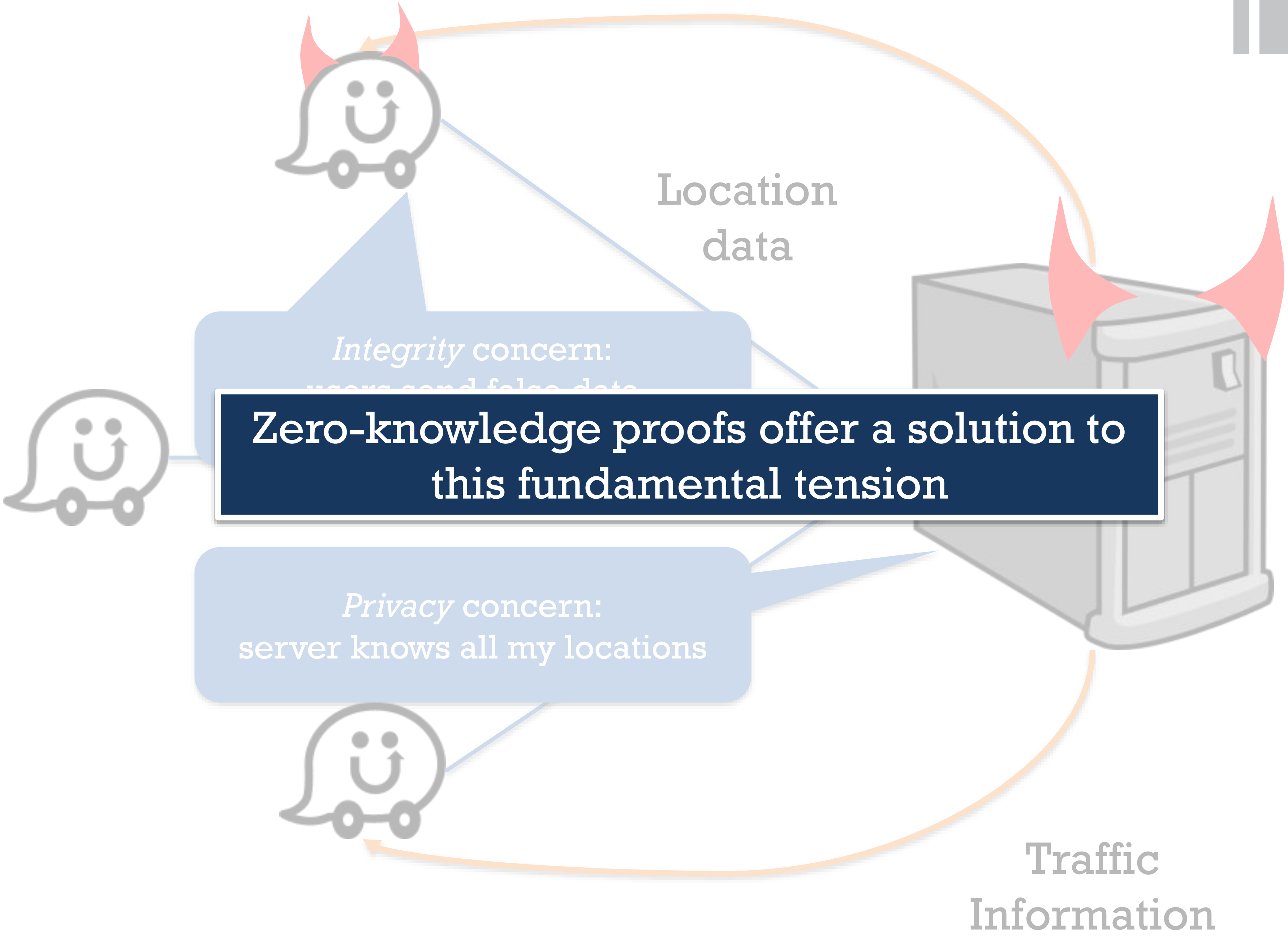


Location data

Integrity concern:
users send false data
to protect their location

Privacy concern:
server knows all my locations

Traffic Information



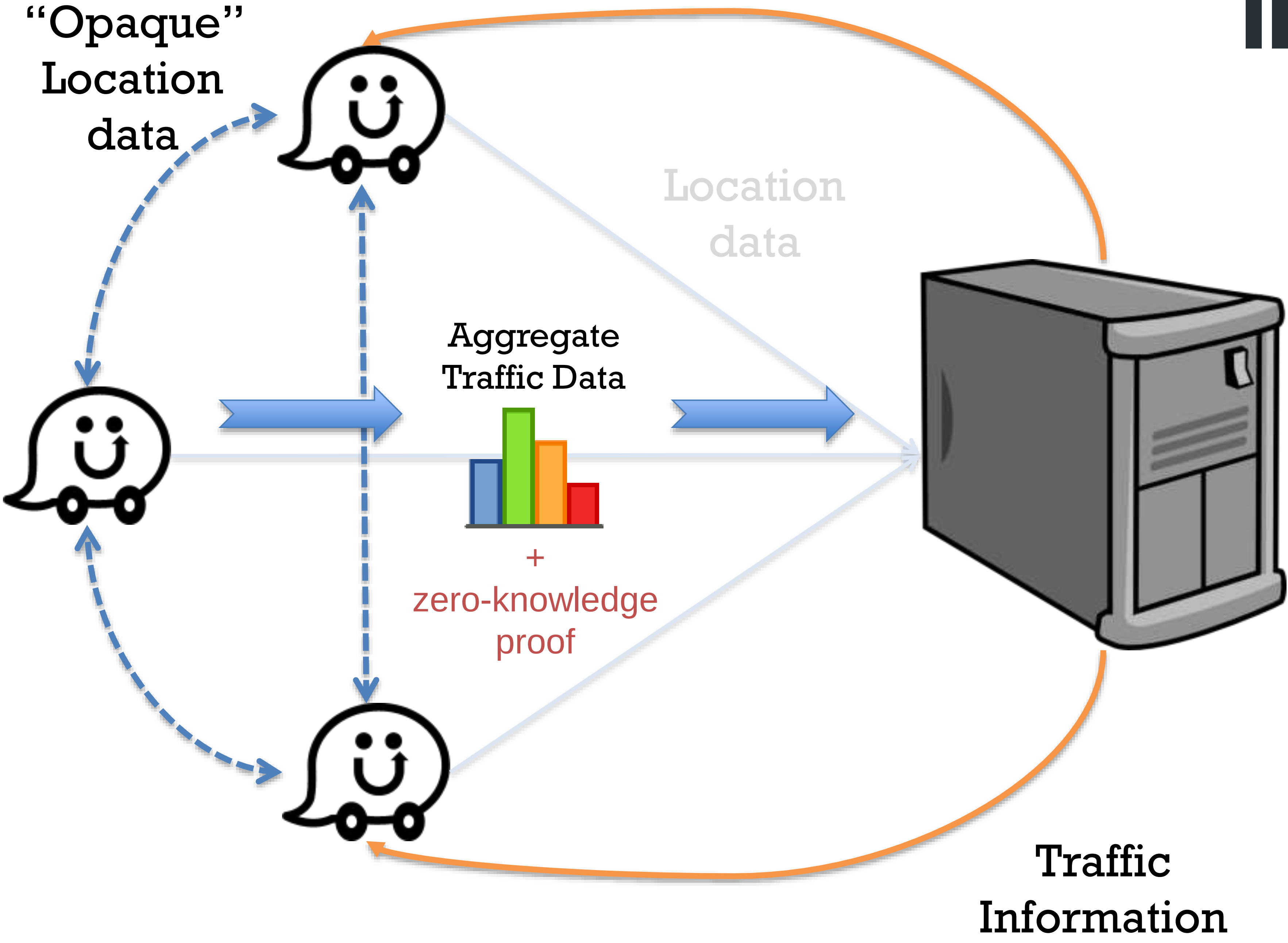
Location data

Integrity concern:
users send false data

Zero-knowledge proofs offer a solution to this fundamental tension

Privacy concern:
server knows all my locations

Traffic Information



"Opaque" Location data

Location data

Aggregate Traffic Data

+ zero-knowledge proof

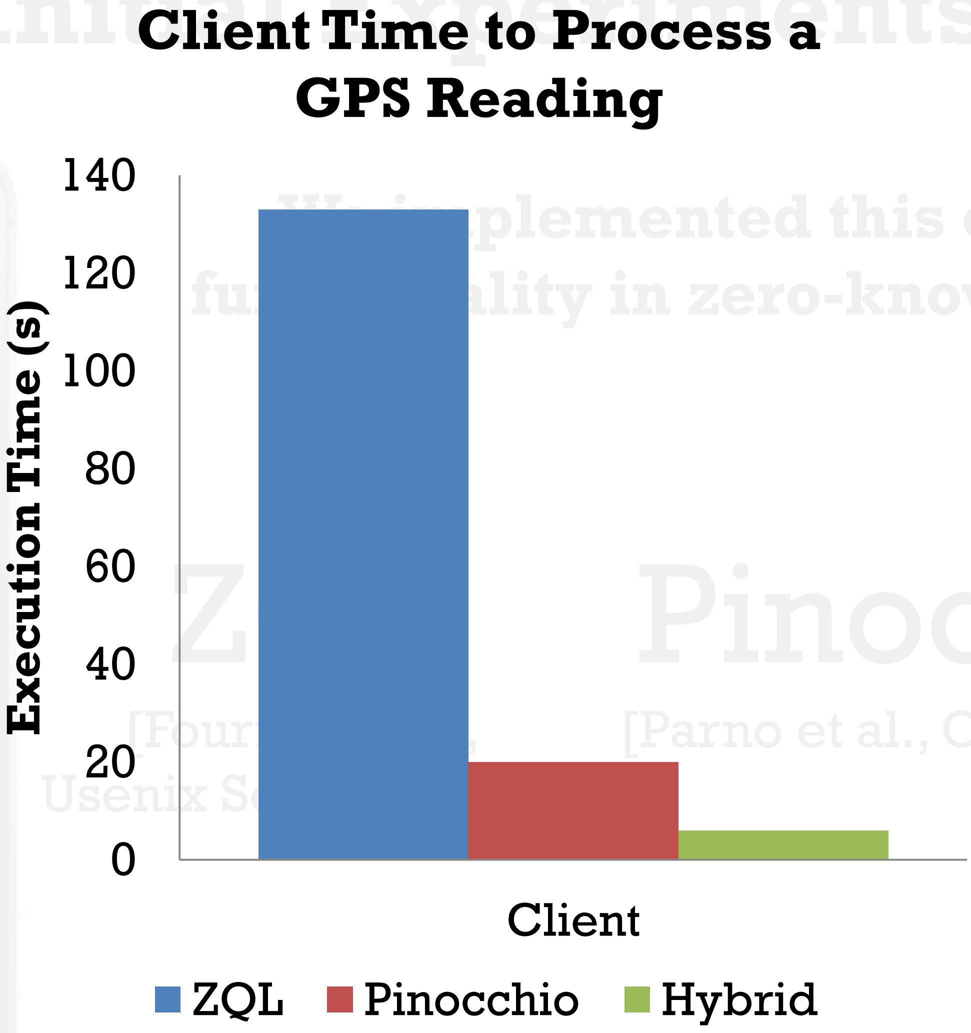
Traffic Information

James Madison Park
Memorial Union
Muir Woods
Observatory Dr
Linden Dr
Bascom Hill
University Ave
MADISON
W Johnson St
Union South
Engineering Dr
Camp Randall Stadium
W Dayton St
Kohl Center
Regent St
N Park St
S Park St
Mickies Dairy Bar
Meriter Hospital
Wisconsin Veterans Museum
Wisconsin State Capitol
Great Dane Pub & Brewing
Monona Terrace
John Nolen Dr
W Washington Ave
E Washington Ave
Williamson St
N Blair St
S Blair St
Proudfit St
N Shore Dr
Brittingham Bay
Monona Bay
Henry Vilas Zoo
VILAS
151
Map data ©2014 Google
Terms of Use
Report a map error
Google

Partition roads into segments

Use *Shamir shares* on segment IDs

Initial Experiments



Why Such a Contrast?

These zero-knowledge "back-ends" have significantly different execution models

ZQL

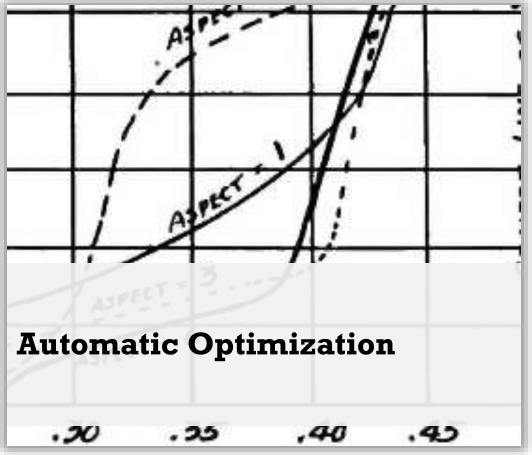
Compiles specialized language to F#, then CIL

Pinocchio

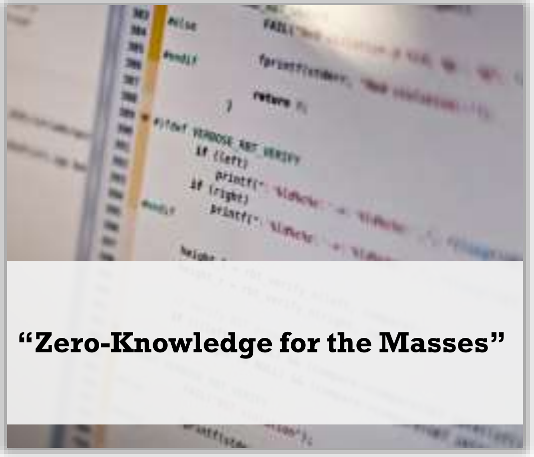
Compiles C to a fixed circuit representation

ZØ: An Optimizing Compiler for ZK

ZØ uses the best of both back-ends as appropriate for the application at hand



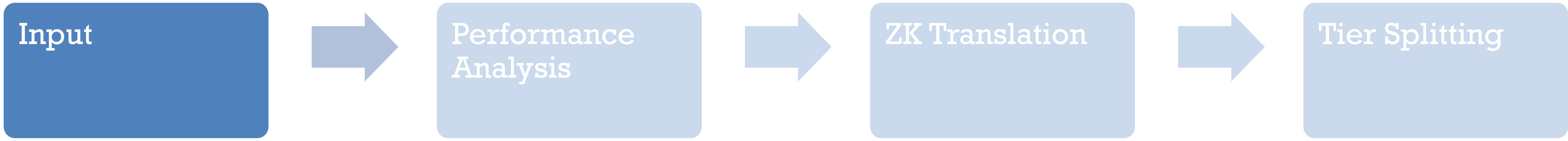
Automatic Optimization



“Zero-Knowledge for the Masses”

Distributed Environments

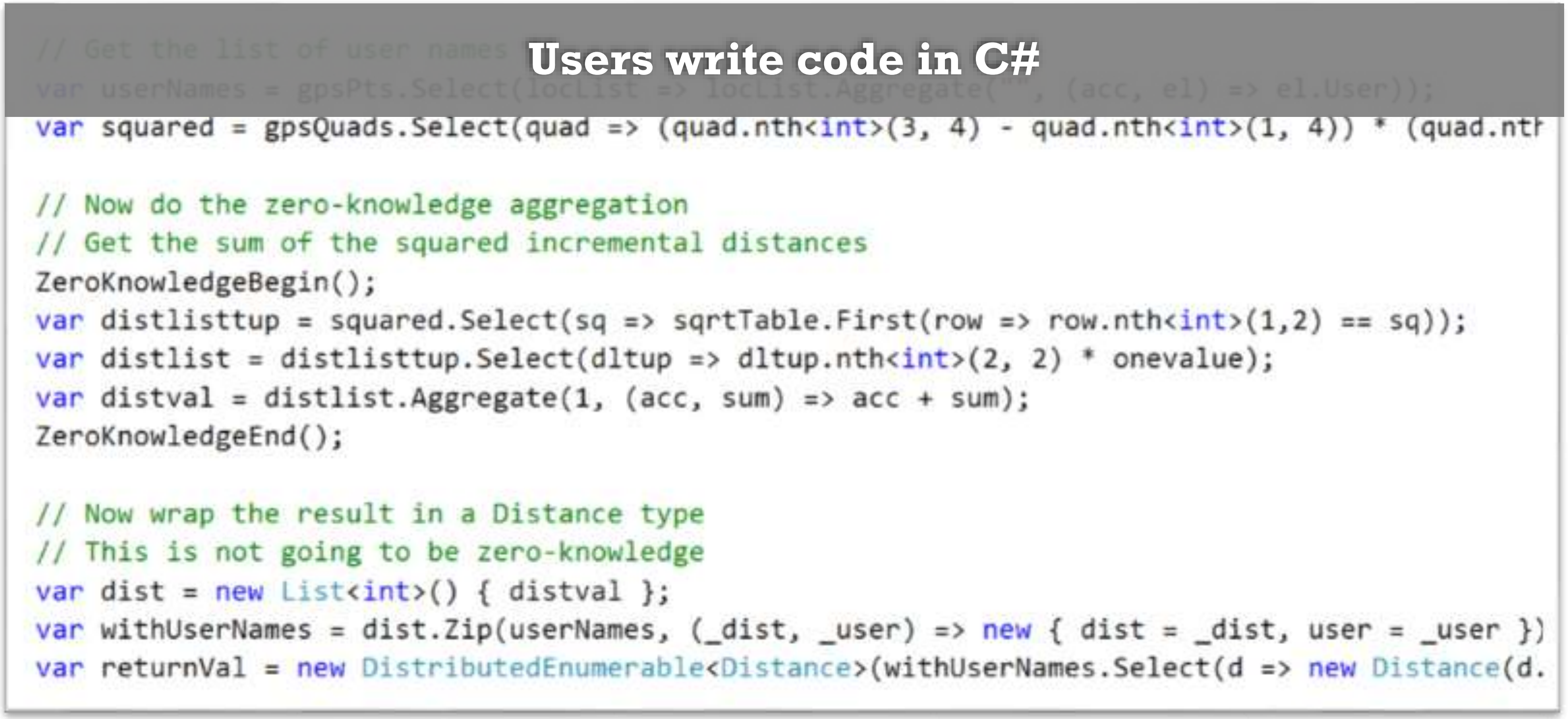
ZØ: An Optimizing Compiler for ZK


Input

Performance Analysis

ZK Translation

Tier Splitting

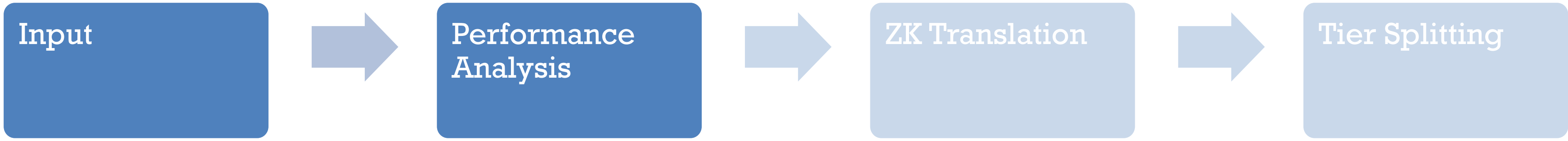

Users write code in C#

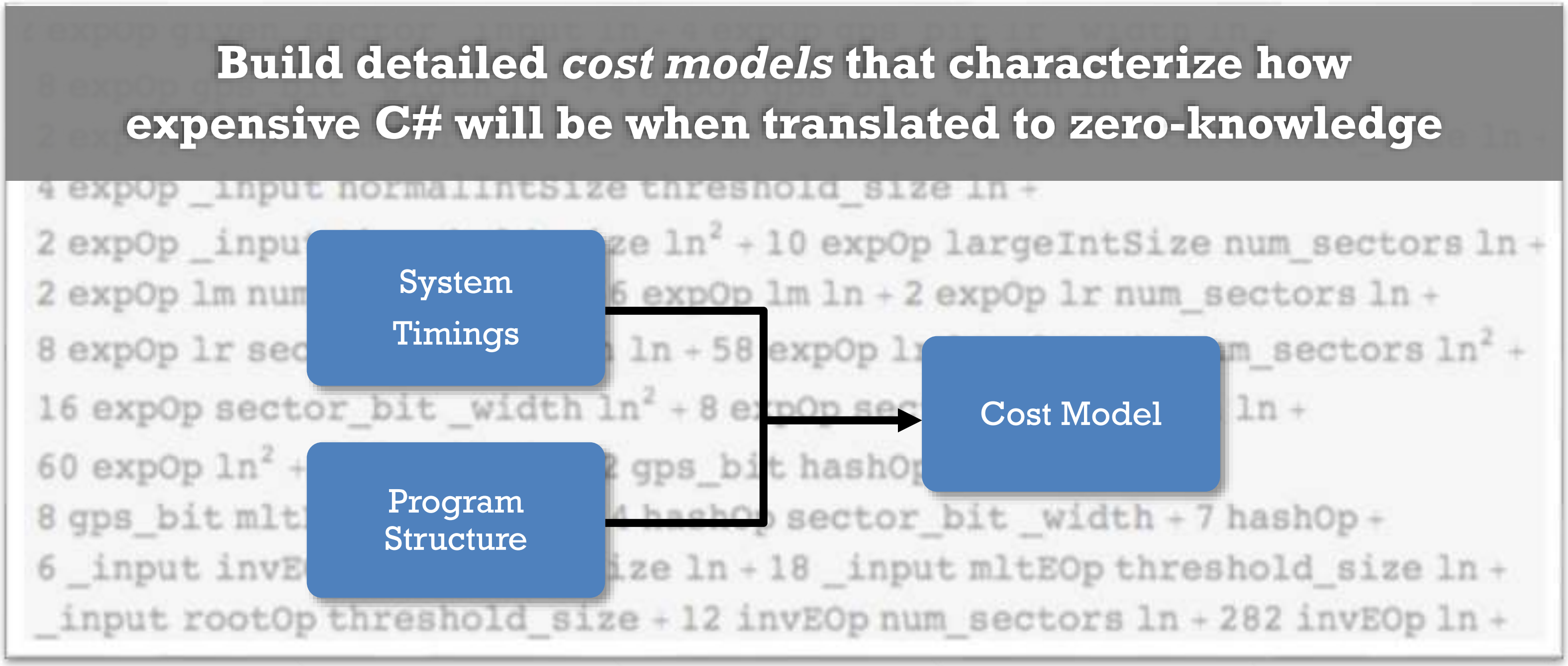
```
// Get the list of user names
var userNames = gpsPts.Select(locList => locList.Aggregate("", (acc, el) => el.User));
var squared = gpsQuads.Select(quad => (quad.nth<int>(3, 4) - quad.nth<int>(1, 4)) * (quad.nth

// Now do the zero-knowledge aggregation
// Get the sum of the squared incremental distances
ZeroKnowledgeBegin();
var distlisttup = squared.Select(sq => sqrtTable.First(row => row.nth<int>(1,2) == sq));
var distlist = distlisttup.Select(dltup => dltup.nth<int>(2, 2) * onevalue);
var distval = distlist.Aggregate(1, (acc, sum) => acc + sum);
ZeroKnowledgeEnd();

// Now wrap the result in a Distance type
// This is not going to be zero-knowledge
var dist = new List<int>() { distval };
var withUserNames = dist.Zip(userNames, (_dist, _user) => new { dist = _dist, user = _user })
var returnVal = new DistributedEnumerable<Distance>(withUserNames.Select(d => new Distance(d.
```

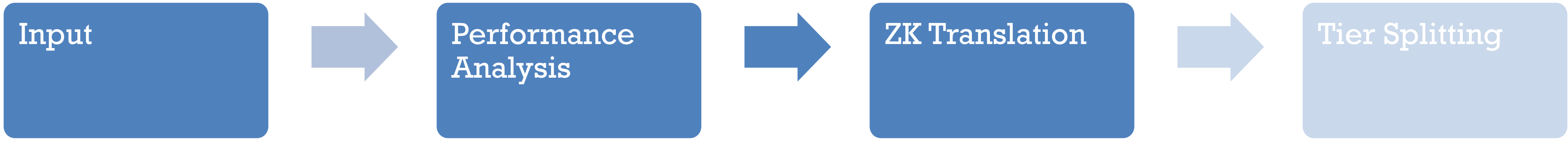
ZØ: An Optimizing Compiler for ZK




Build detailed *cost models* that characterize how expensive C# will be when translated to zero-knowledge

System Timings
Program Structure
Cost Model
4 expOp _input normalIntSize threshold_size ln +
2 expOp _inpu … ze ln² + 10 expOp largeIntSize num_sectors ln +
2 expOp lm num … 6 expOp lm ln + 2 expOp lr num_sectors ln +
8 expOp lr sec … ln + 58 expOp l … m_sectors ln² +
16 expOp sector_bit_width ln² + 8 expOp sec … ln +
60 expOp ln² + … 2 gps_bit hashOp
8 gps_bit mlt … 4 hashOp sector_bit_width + 7 hashOp +
6 _input invE … ize ln + 18 _input mltEOp threshold_size ln +
_input rootOp threshold_size + 12 invEOp num_sectors ln + 282 invEOp ln +

ZØ: An Optimizing Compiler for ZK

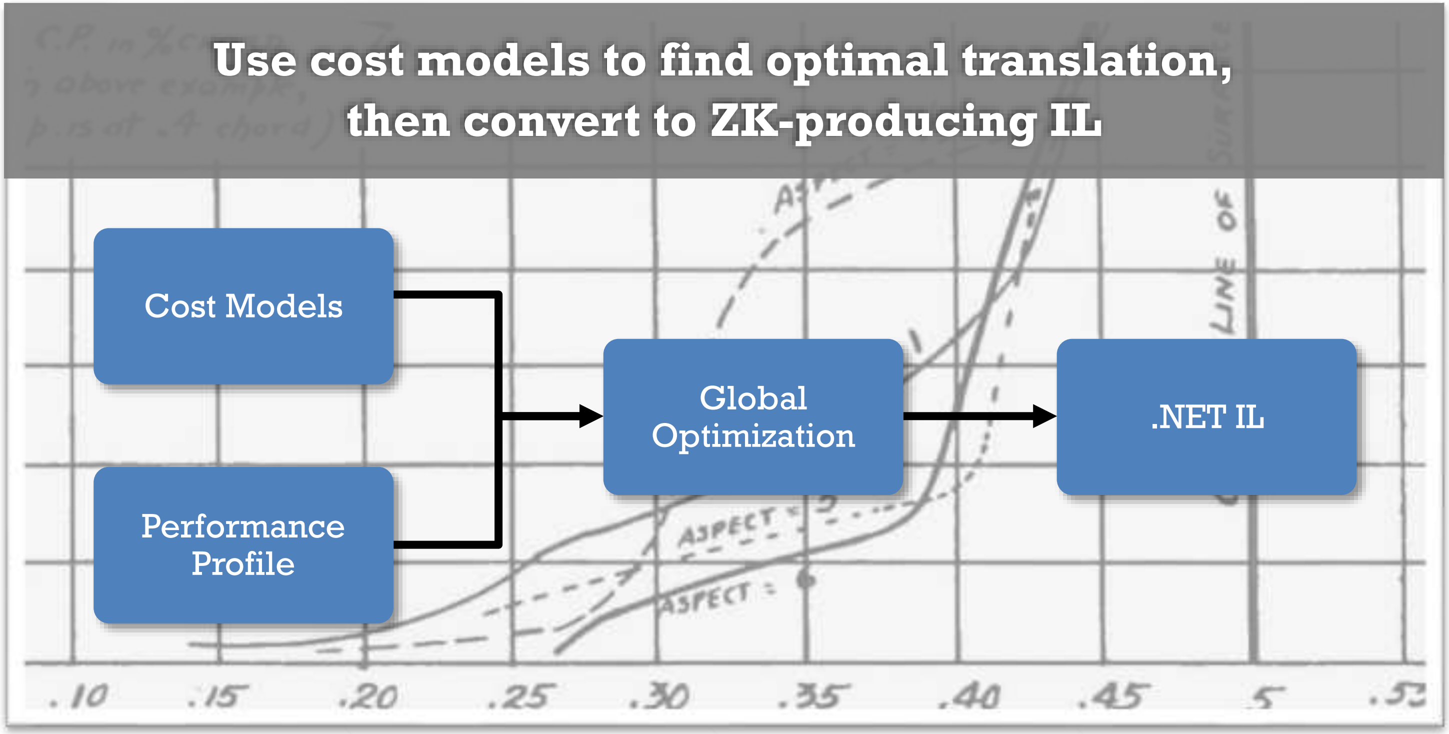


Input

Performance Analysis

ZK Translation

Tier Splitting



Use cost models to find optimal translation, then convert to ZK-producing IL

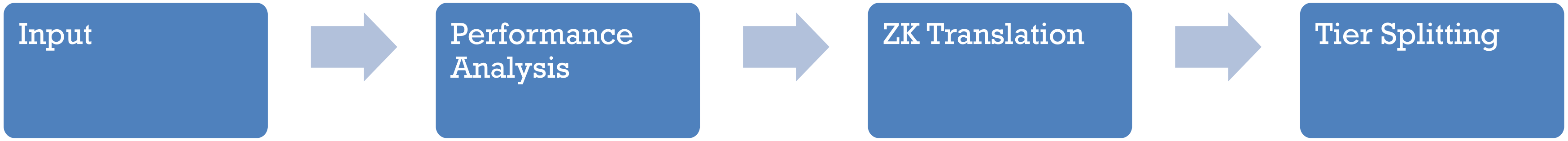
Cost Models

Performance Profile

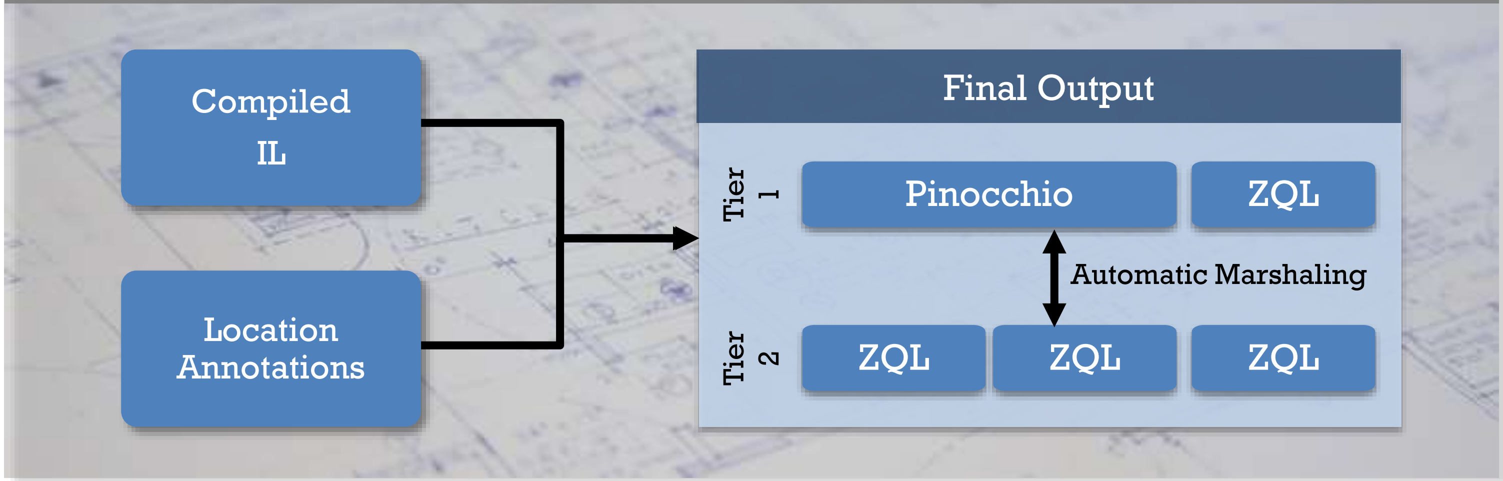
Global Optimization

.NET IL

ZØ: An Optimizing Compiler for ZK



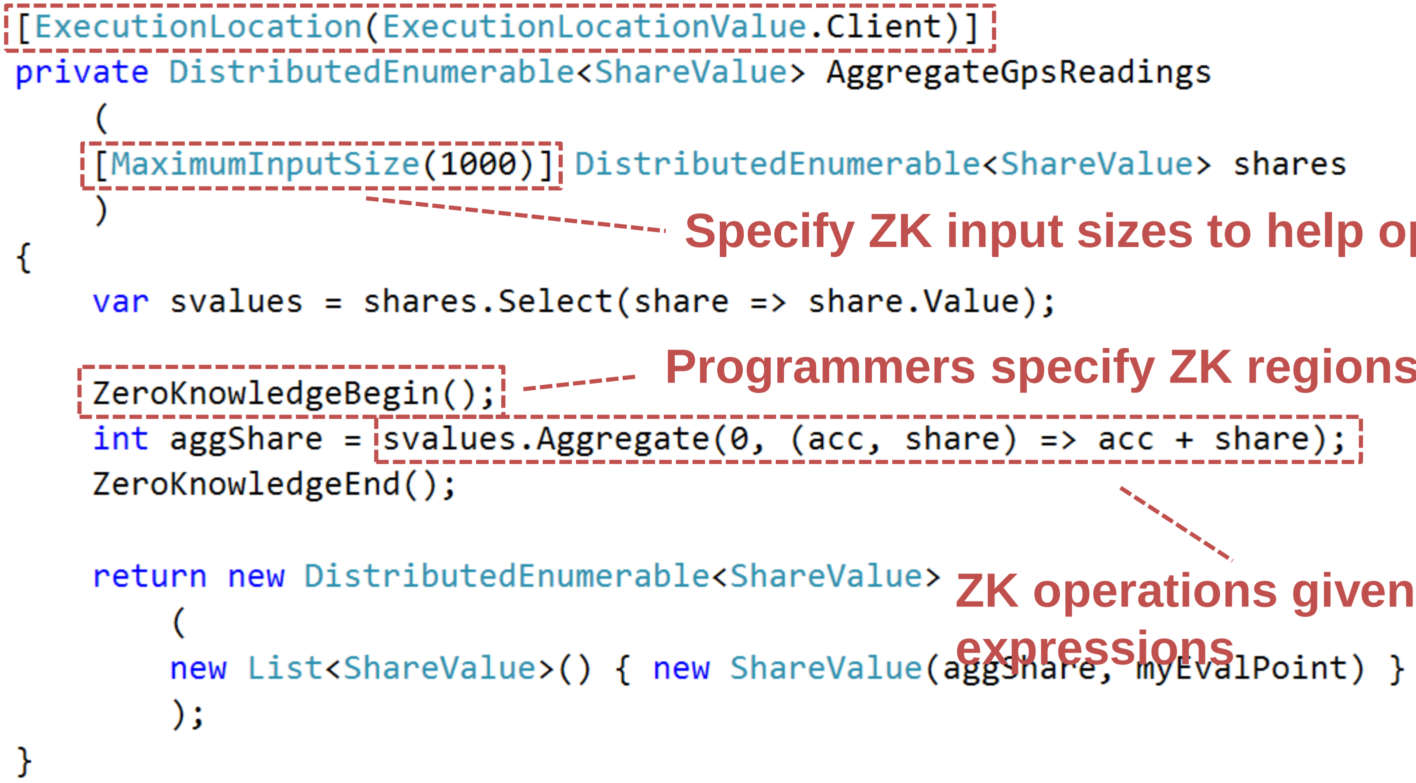
Use location annotations to split IL between tiers, insert automatic data transfer and synchronization



ZERO-KNOWLEDGE IN C#

Zero-Knowledge in C#

Location annotations drive tier-splitting



```
[ExecutionLocation(ExecutionLocationValue.Client)]
private DistributedEnumerable<ShareValue> AggregateGpsReadings
    (
    [MaximumInputSize(1000)] DistributedEnumerable<ShareValue> shares
    )
{
    var svalues = shares.Select(share => share.Value);

    ZeroKnowledgeBegin();
    int aggShare = svalues.Aggregate(0, (acc, share) => acc + share);
    ZeroKnowledgeEnd();

    return new DistributedEnumerable<ShareValue>
        (
        new List<ShareValue>() { new ShareValue(aggShare, myEvalPoint) }
        );
}
```

Specify ZK input sizes to help optimization

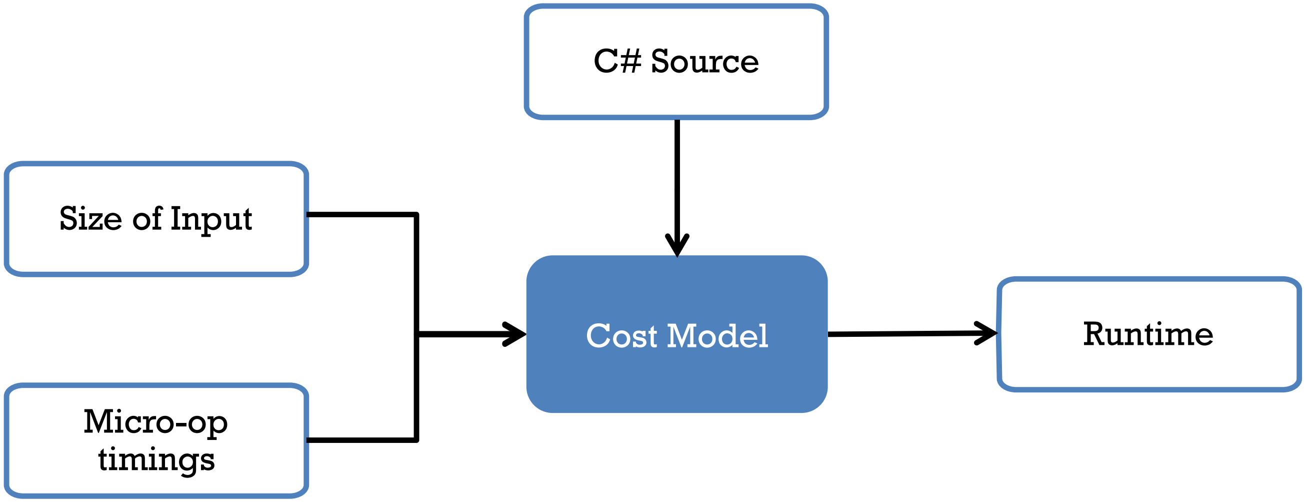
Programmers specify ZK regions

ZK operations given by LINQ expressions

COST MODELING

Cost Models for Optimization

Cost models characterize the ZK runtime of C# code



$$\mathbf{F}(\mathit{inputListSize}) = \mathrm{eqOp} * \underbrace{\boxed{\mathit{inputListSize}}}_{\text{size of input}} + \underbrace{\boxed{\mathrm{addOp}}}_{\text{micro-op timings}} + 12*\mathrm{expOp} + 3 * \mathrm{extendOp} + 14*\mathrm{mltOp}$$

Building a Cost Model

Symbolic evaluation over polynomial domain

ZQL

`map`, `fold`, `find` expressions: we can always bound the number of ops in each expression

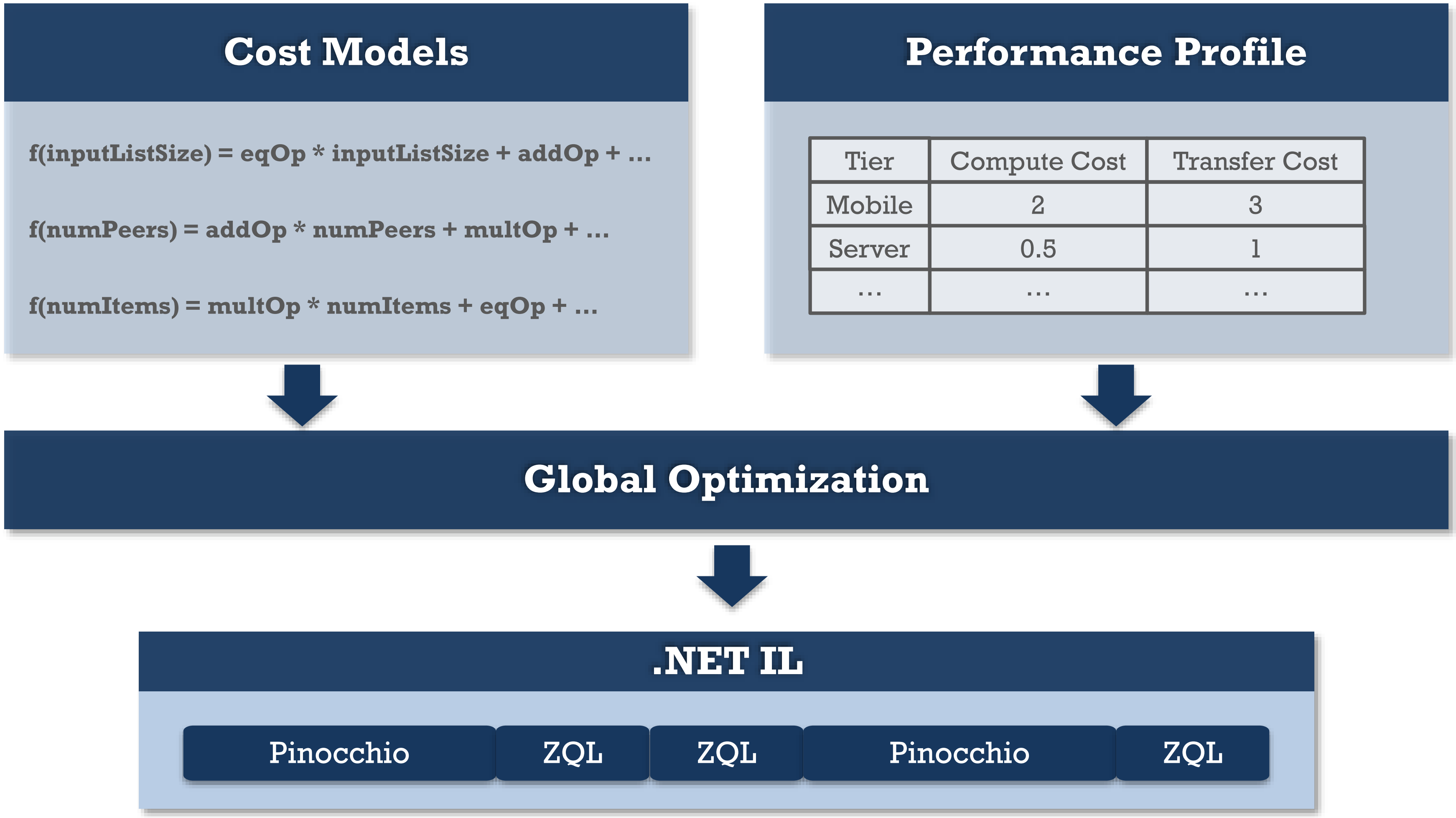
Static circuit evaluation polynomials

Pinocchio

Given a circuit, we can determine evaluation and proof generation time

TRANSLATION & TIER SPLITTING

Translating C# to Zero-Knowledge



Cost Models

f(inputListSize) = eqOp * inputListSize + addOp + …

f(numPeers) = addOp * numPeers + multOp + …

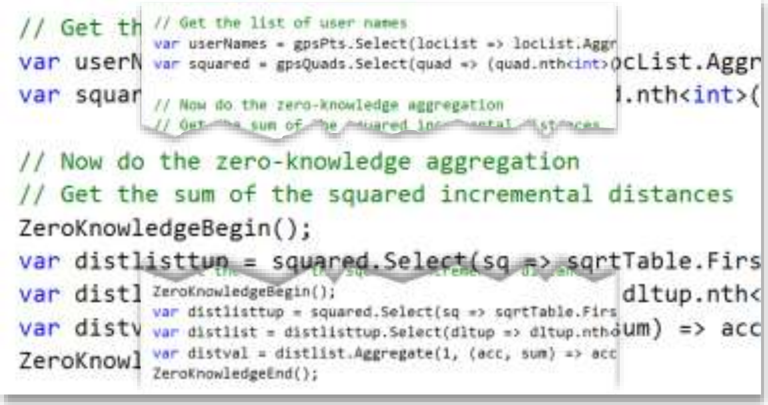
f(numItems) = multOp * numItems + eqOp + …

Performance Profile

Tier	Compute Cost	Transfer Cost
Mobile	2	3
Server	0.5	1
…	…	…

Global Optimization

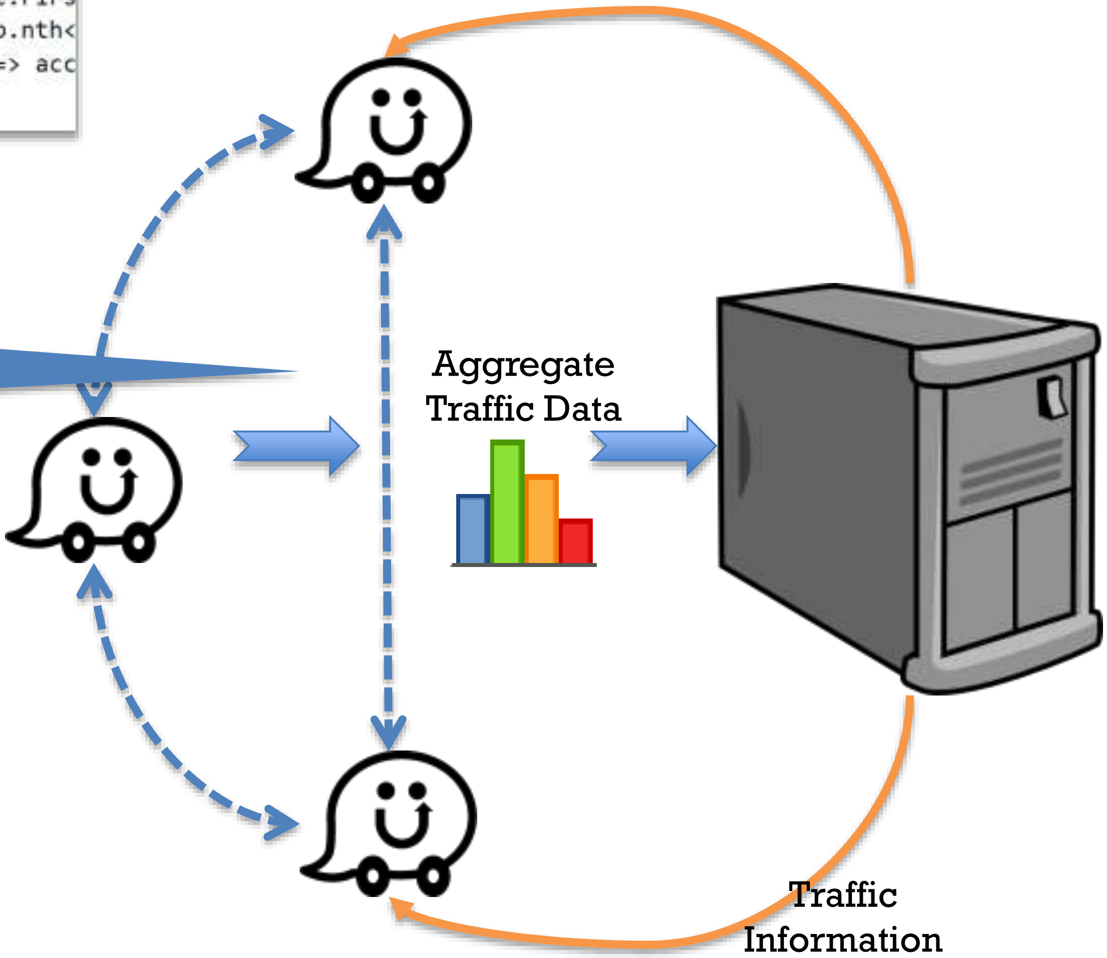
.NET IL

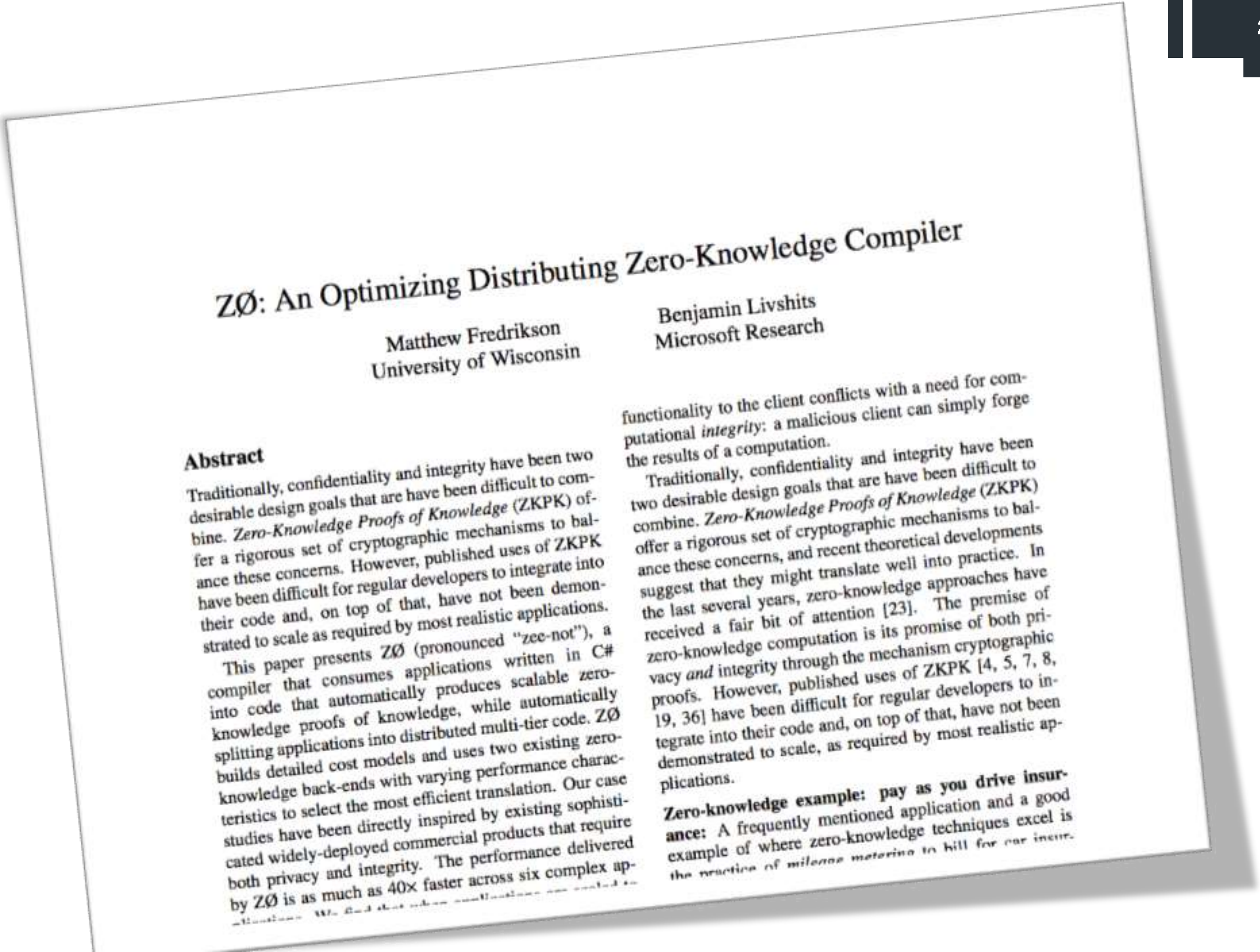


```
// Get the list of user names
var userNames = gpsPts.Select(locList => locList.Aggr
var squared = gpsQuads.Select(quad => (quad.nth<int>(

// Now do the zero-knowledge aggregation
// Get the sum of the squared incremental distances
ZeroKnowledgeBegin();
var distlisttup = squared.Select(sq => sqrtTable.Firs
var distlist = distlisttup.Select(dltup => dltup.nth<
var distval = distlist.Aggregate(1, (acc, sum) => acc
ZeroKnowledgeEnd();
```

Insert code for marshaling and synchronization





ZØ: An Optimizing Distributing Zero-Knowledge Compiler

Matthew Fredrikson
University of Wisconsin

Benjamin Livshits
Microsoft Research

Abstract

Traditionally, confidentiality and integrity have been two desirable design goals that are have been difficult to combine. *Zero-Knowledge Proofs of Knowledge* (ZKPK) offer a rigorous set of cryptographic mechanisms to balance these concerns. However, published uses of ZKPK have been difficult for regular developers to integrate into their code and, on top of that, have not been demonstrated to scale as required by most realistic applications.

This paper presents ZØ (pronounced "zee-not"), a compiler that consumes applications written in C# into code that automatically produces scalable zero-knowledge proofs of knowledge, while automatically splitting applications into distributed multi-tier code. ZØ builds detailed cost models and uses two existing zero-knowledge back-ends with varying performance characteristics to select the most efficient translation. Our case studies have been directly inspired by existing sophisticated widely-deployed commercial products that require both privacy and integrity. The performance delivered by ZØ is as much as 40× faster across six complex applications. We find that when applications are scaled to

functionality to the client conflicts with a need for computational *integrity*: a malicious client can simply forge the results of a computation.

Traditionally, confidentiality and integrity have been two desirable design goals that are have been difficult to combine. *Zero-Knowledge Proofs of Knowledge* (ZKPK) offer a rigorous set of cryptographic mechanisms to balance these concerns, and recent theoretical developments suggest that they might translate well into practice. In the last several years, zero-knowledge approaches have received a fair bit of attention [23]. The premise of zero-knowledge computation is its promise of both privacy *and* integrity through the mechanism cryptographic proofs. However, published uses of ZKPK [4, 5, 7, 8, 19, 36] have been difficult for regular developers to integrate into their code and, on top of that, have not been demonstrated to scale, as required by most realistic applications.

Zero-knowledge example: pay as you drive insurance: A frequently mentioned application and a good example of where zero-knowledge techniques excel is the practice of *mileage metering* to bill for car insur-

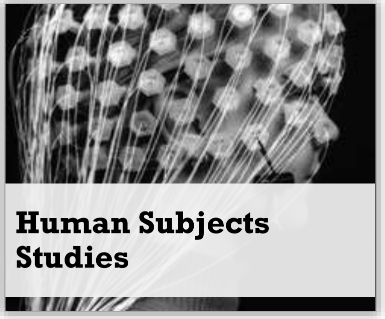
EVALUATION

Experiments

We ran each application in three configurations

ZQL

Crowd-sourced traffic maps


Human Subjects Studies

Pinocchio

Personal Fitness Rewards

Collaborative Recommender

ZØ

Retail Loyalty Card

Collaborative NIDS

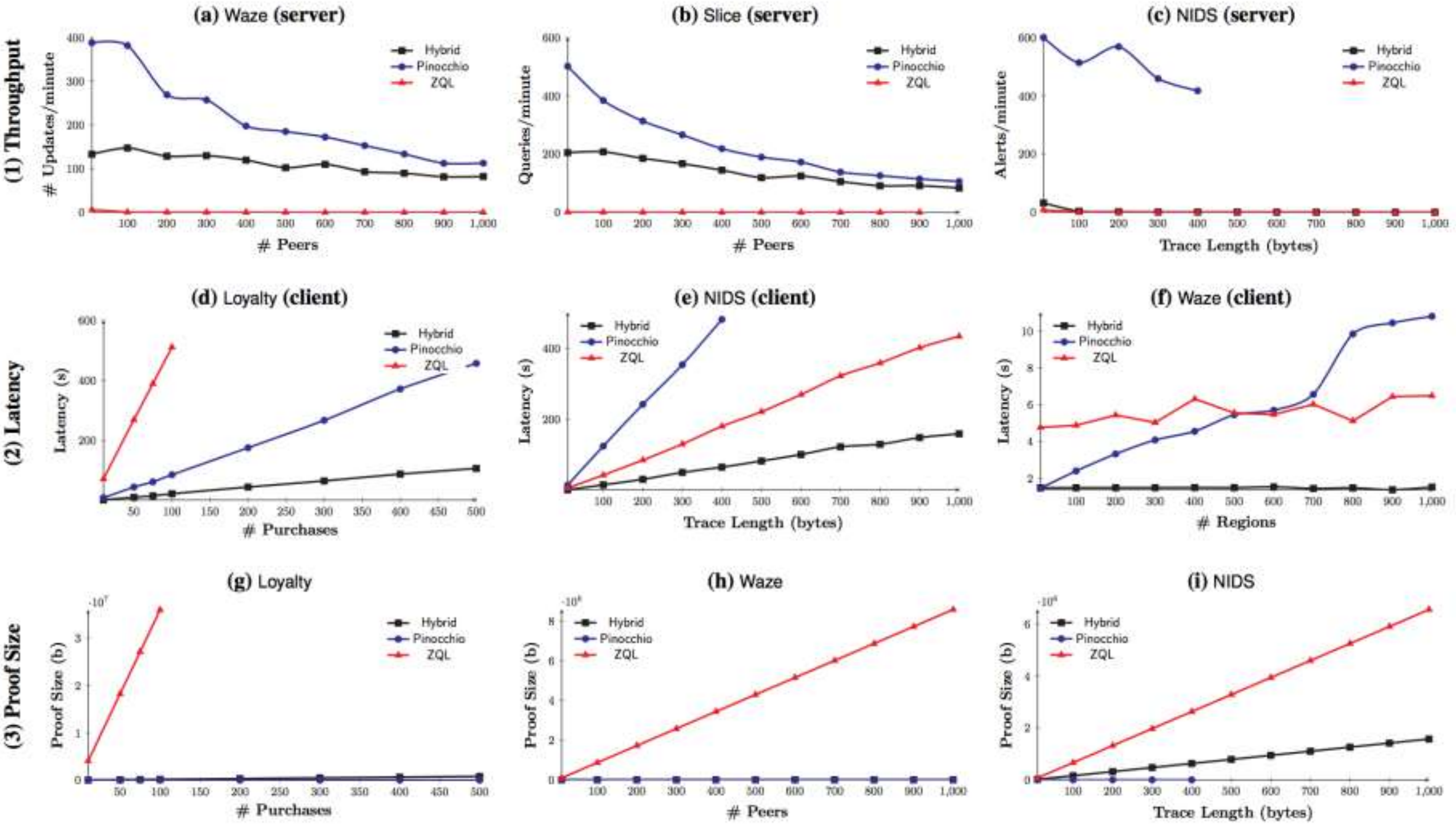
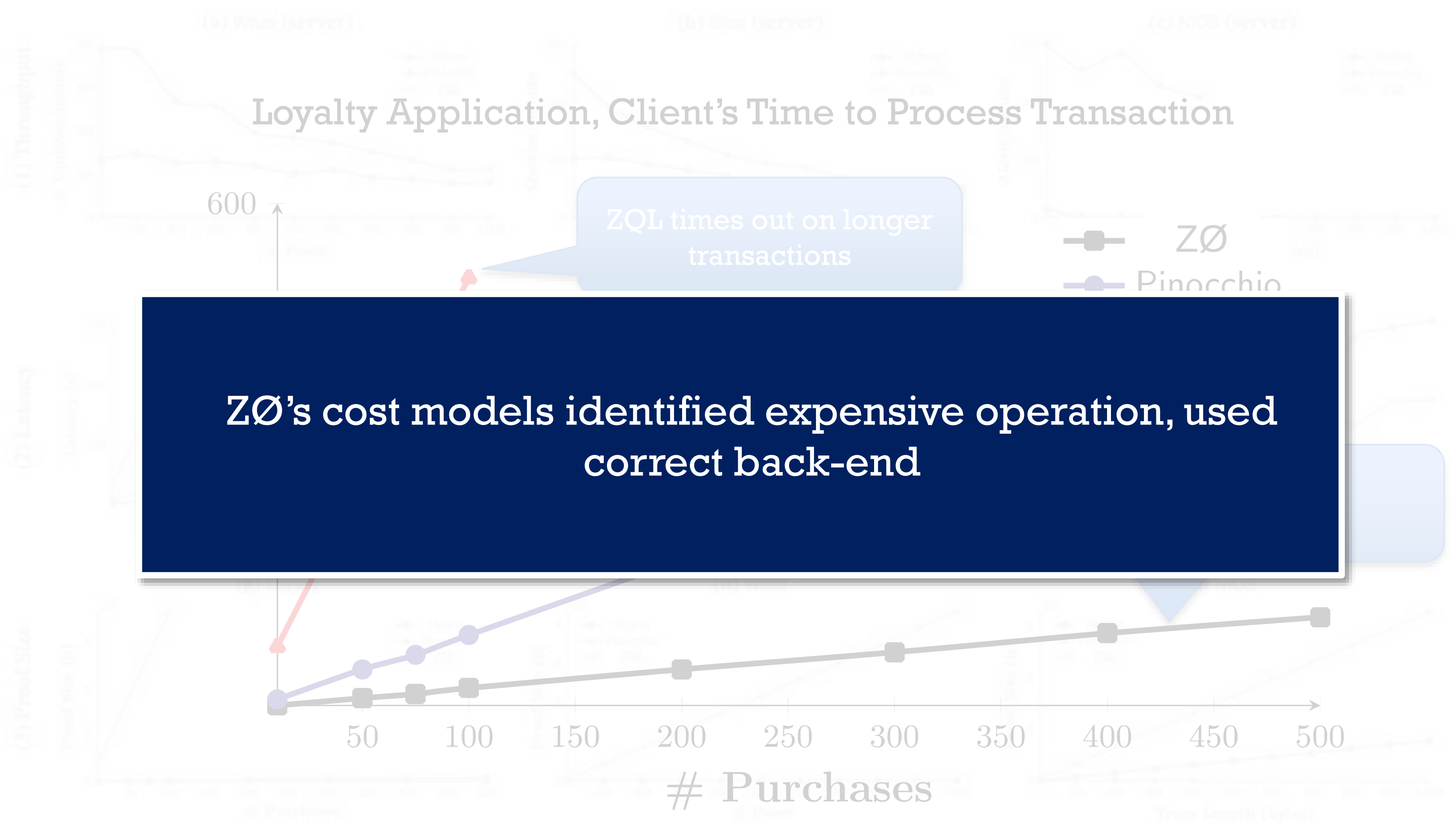
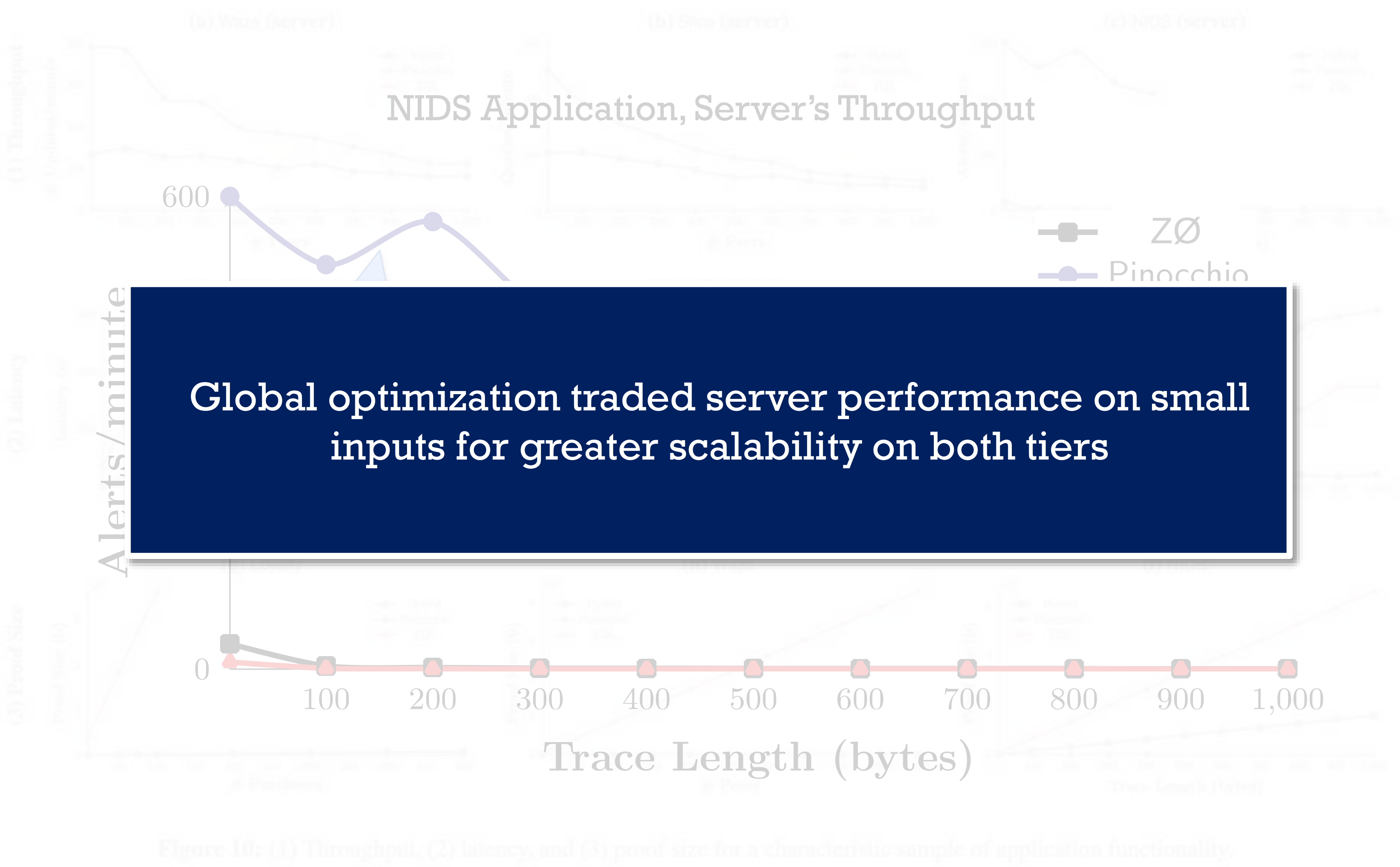


Figure 10: (1) Throughput, (2) latency, and (3) proof size for a characteristic sample of application functionality.



ZØ's cost models identified expensive operation, used correct back-end

Figure 10: (1) Throughput, (2) latency, and (3) proof size for a characteristic sample of application functionality.



Global optimization traded server performance on small inputs for greater scalability on both tiers

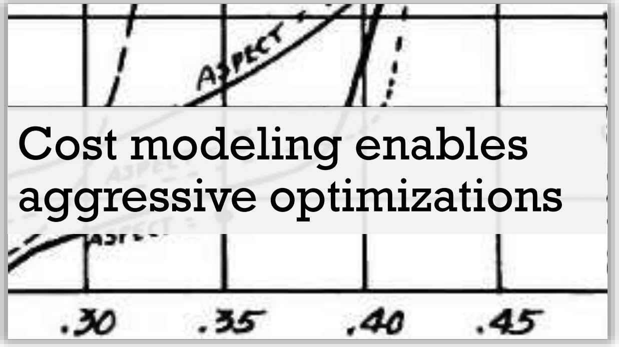
Experiments

We ran each application in three configurations

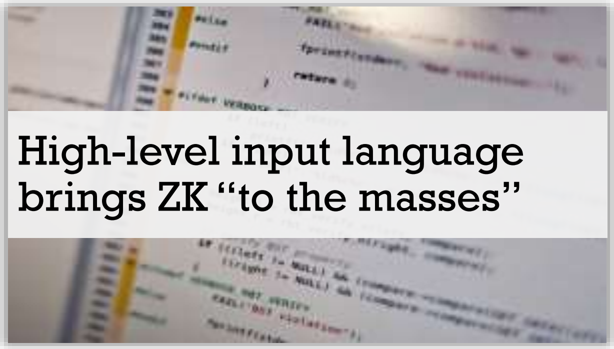
ZQL Pinocchio ZØ

Scaling	Scales up to 10x larger data
Performance	Up to 40x improvement in runtime
Proof Size	Up to 10-100x smaller than ZQL

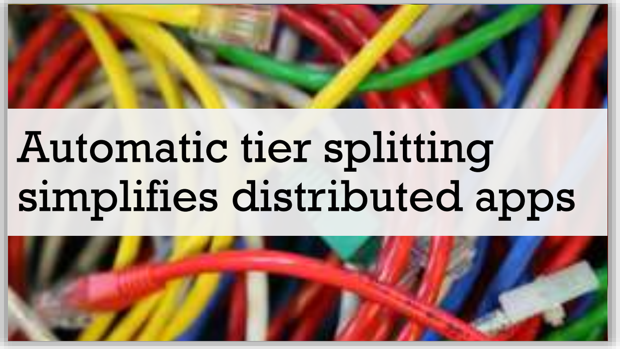
Conclusions



Cost modeling enables aggressive optimizations



High-level input language brings ZK “to the masses”

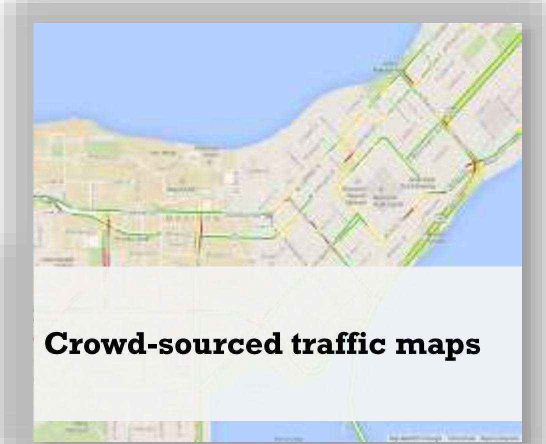


Automatic tier splitting simplifies distributed apps

Illustrated benefits with six applications

Backup Slides

This talk: at a glance

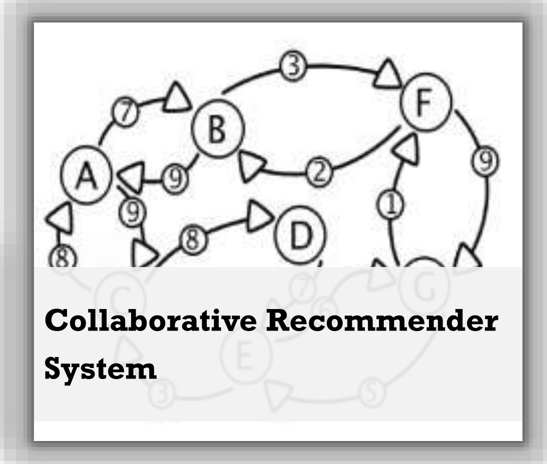


Crowd-sourced traffic maps

Personal Fitness Rewards

Retail Loyalty Card

Human Subjects Studies



Collaborative Recommender System

Collaborative NIDS

Conclusions

- ZØ is a new zero-knowledge compiler
  - Detailed cost modeling enable aggressive optimizations
  - High-level language brings ZK "to the masses"
  - Automatic tier splitting simplifies distributed apps

- Illustrated benefits with six interesting apps
  - ZØ's optimizations make these feasible

Thanks!

Modern apps demand personal data

Often the need for data is legitimate

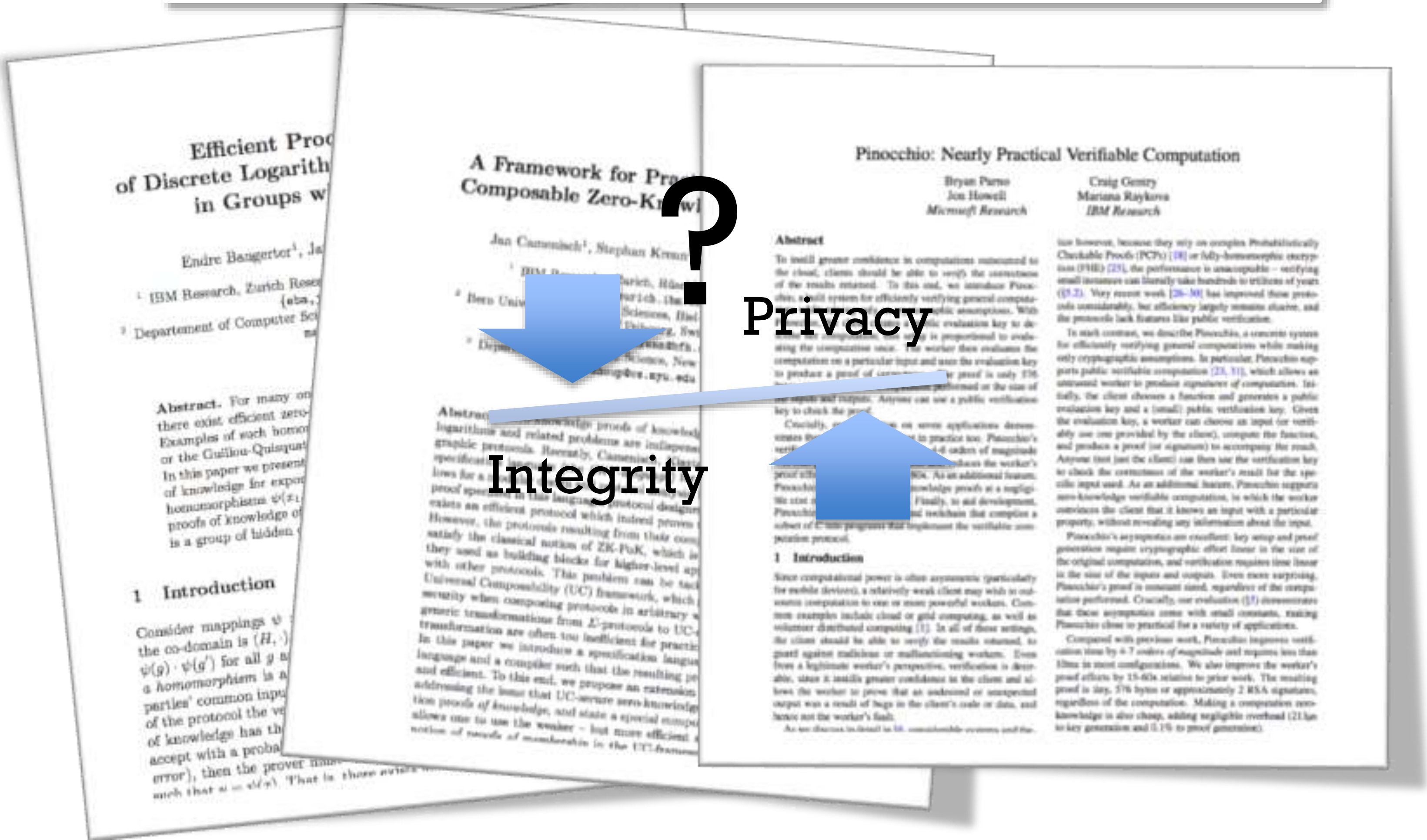
Pressure to address privacy concerns is widespread

In many applications, this creates a tension between *privacy* and *integrity*

Zero-Knowledge: A Promising Solution

Batsvef

Prove that a computation was performed correctly without revealing inputs



Efficient Proo… of Discrete Logarith… in Groups w…

A Framework for Pra… Composable Zero-Kn…

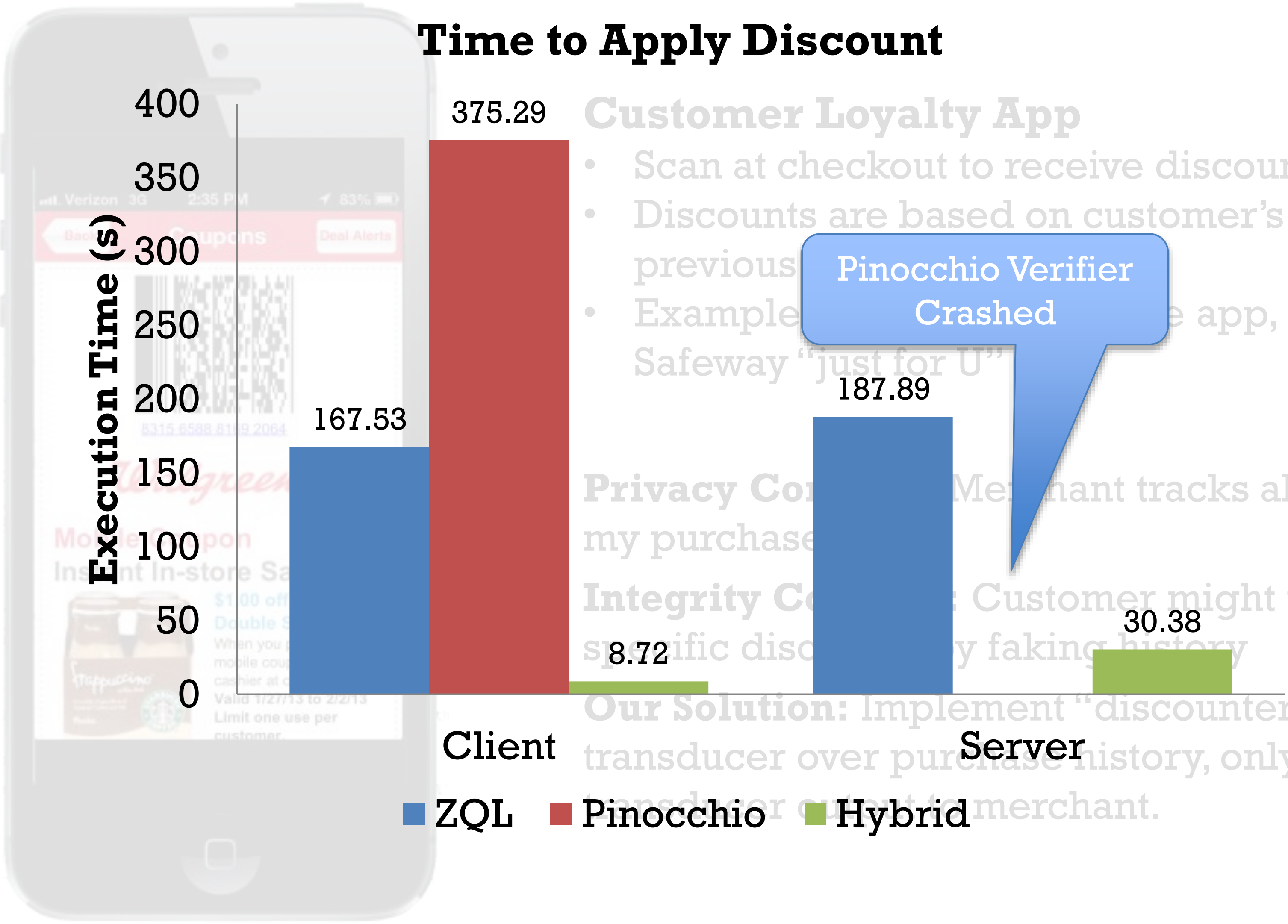
Pinocchio: Nearly Practical Verifiable Computation

- The map is broken into regions, and the desired statistic is the number of clients in each region at time *t.*
- At regular intervals, the server requests density stats from the clients.
- On receiving a request, each client:

Zero-Knowledge

1. Takes a GPS reading
2. Computes its map region
3. Encodes its region as a vector, zero everywhere but the column for its region
4. Creates shares of its vector, sends them to other clients
5. On receiving the other clients' shares, each client sums all received shares and sends the result to the server

- On receiving the summed shares from the clients, the server reconstructs the sum to obtain the density map



Pinocchio Verifier Crashed

Customer Loyalty App

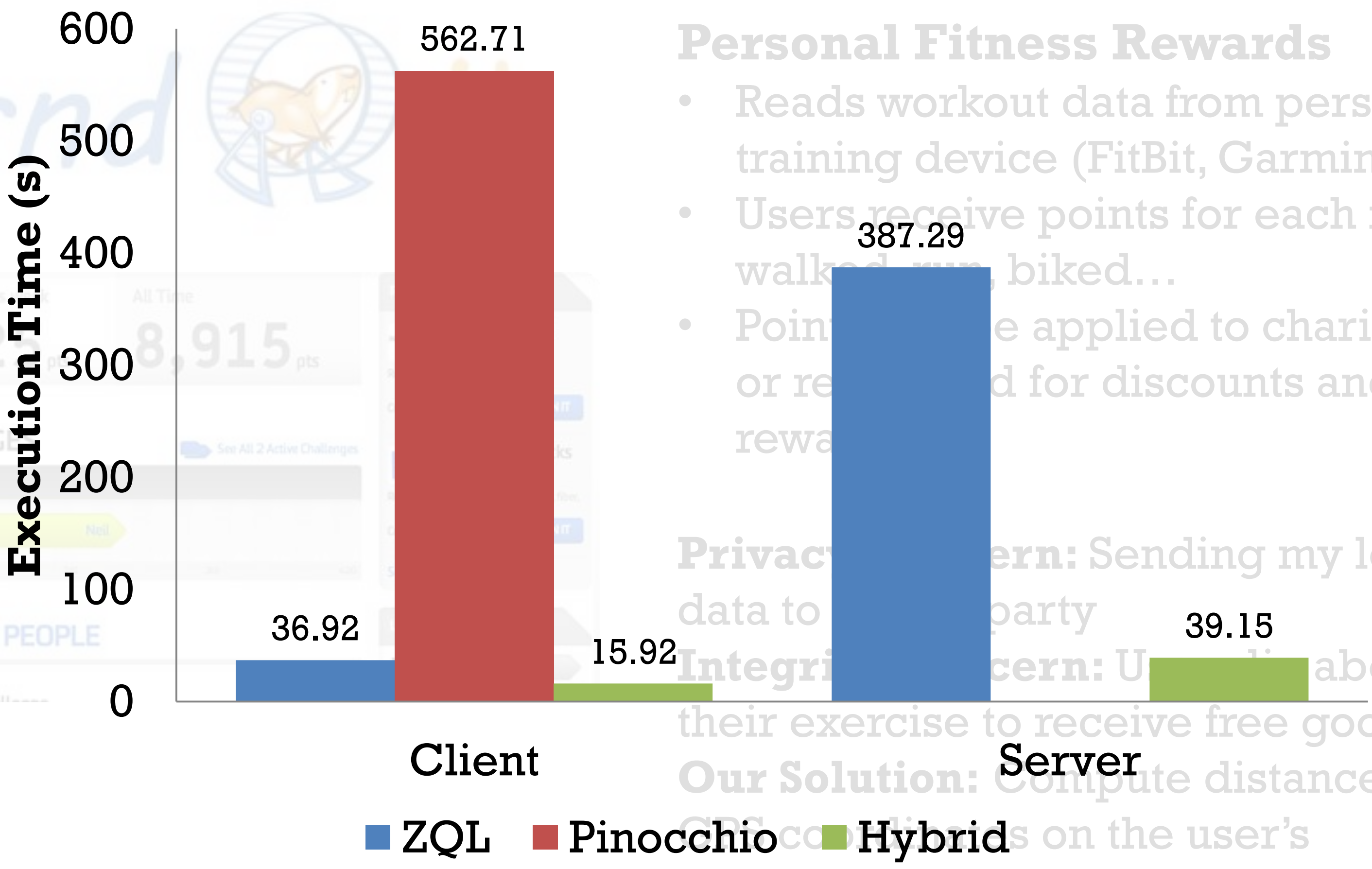
- Scan at checkout to receive discounts
- Discounts are based on customer's previous
- Example e app, Safeway "just for U"

Privacy Co Merchant tracks all of my purchase

Integrity Co : Customer might target specific disc y faking history

Our Solution: Implement "discounter" as transducer over purchase history, only send transducer output to merchant.

Time to Redeem Workout



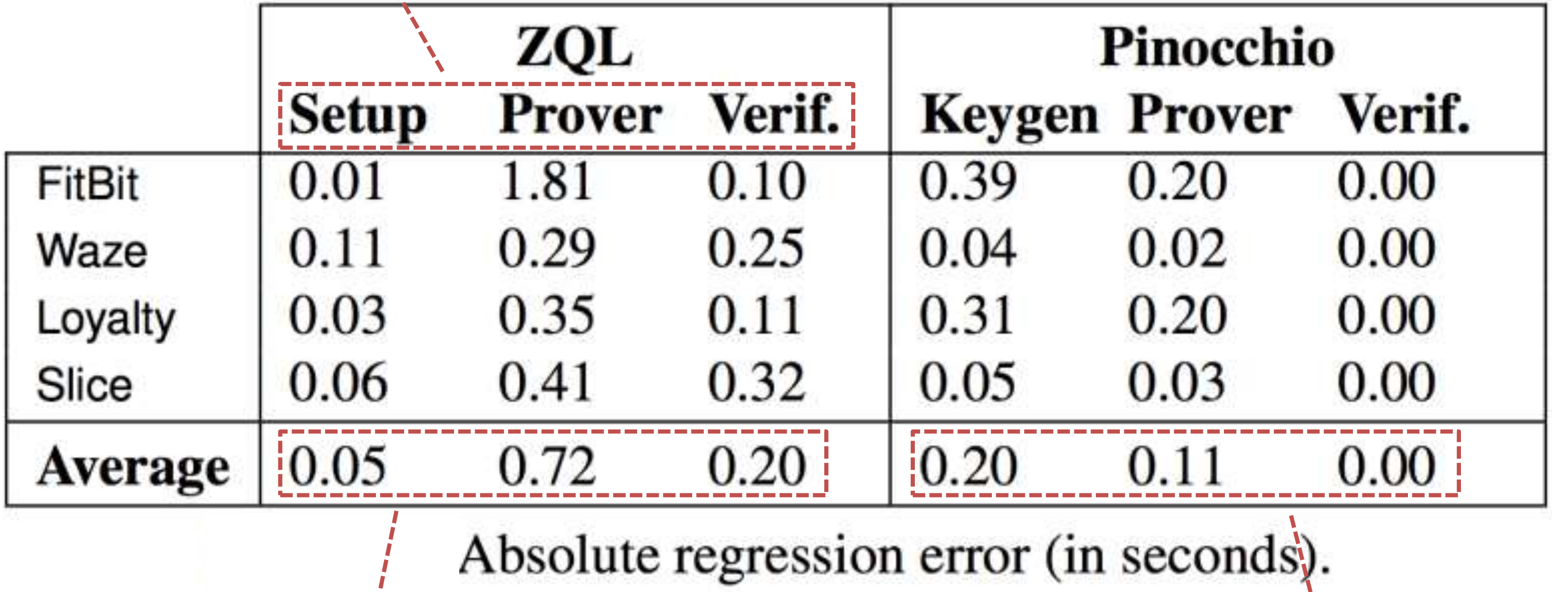
Personal Fitness Rewards

- Reads workout data from personal training device (FitBit, Garmin, …)
- Users receive points for each mile walk , biked…
- Poin e applied to charities, or re d for discounts and rewa

Privac ern: Sending my location data to party
Integri cern: U about their exercise to receive free goods
Our Solution: Compute distance from GPS co s on the user's computer, send final result to third party

Cost Model Accuracy

Different stages of a single ZK computation



	ZQL			Pinocchio		
	Setup	Prover	Verif.	Keygen	Prover	Verif.
FitBit	0.01	1.81	0.10	0.39	0.20	0.00
Waze	0.11	0.29	0.25	0.04	0.02	0.00
Loyalty	0.03	0.35	0.11	0.31	0.20	0.00
Slice	0.06	0.41	0.32	0.05	0.03	0.00
Average	0.05	0.72	0.20	0.20	0.11	0.00

Absolute regression error (in seconds).

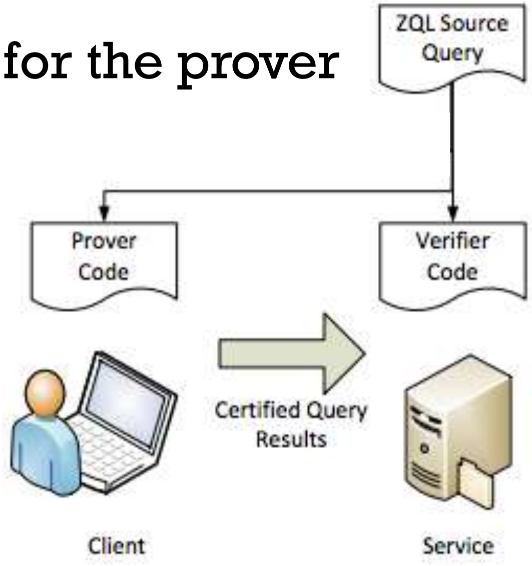
0.32 seconds on average (9%)

0.1 seconds on average (14%)

ZQL

Target code is purely-functional, operates on F# lists

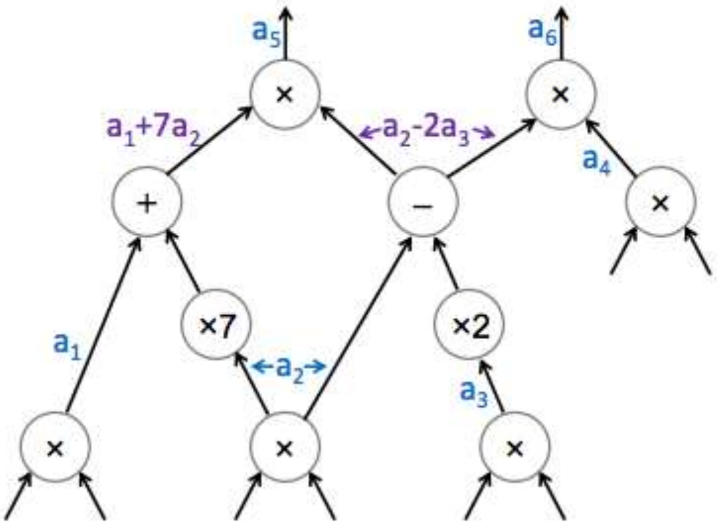
- Translated code mimics structure of original program, does additional cryptographic work for each primitive operation
- Relies heavily on a few primitive operations: `map`, `fold`, `find`
- Lambdas allowed only in limited contexts
- Translated code is highly parallelizable, esp. for the prover
- Runtime available for WP 7 and 8



Pinocchio

Target code is a fixed-length arithmetic circuit

- Input language is C with static loops, constant dereferences, no recursion
- Everything is in-lined
- Values are broken into constituent bits, Boolean operations used
- Circuit evaluator/prover is optimized native code
- Requires polynomial interpolation and division
- No support for parallel execution


a_5
a_6
×
×
a_1+7a_2
$\leftarrow a_2-2a_3 \rightarrow$
a_4
+
−
×
×7
×2
a_1
$\leftarrow a_2 \rightarrow$
a_3
×
×
×

Goals for ZØ

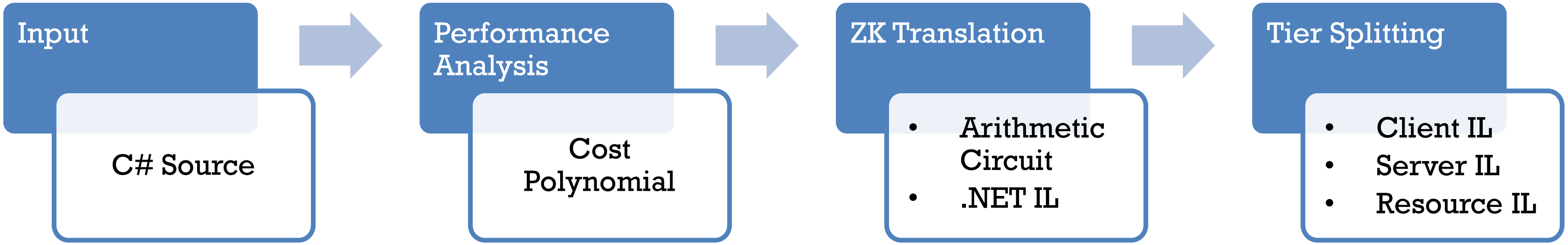
Performance

- Neither back-end is one-size-fits-all
- Understanding performance requires specialized knowledge
- Bring zero-knowledge to "the masses"

Usability

- Users should *never write their own crypto*
- Seamless integration with existing code
  - LINQ is our bridge to zero-knowledge
  - Can integrate ZK with large amounts of UI, Libraries, arbitrary logic
- Automates tier-splitting

ZØ: An Optimizing Compiler for ZK



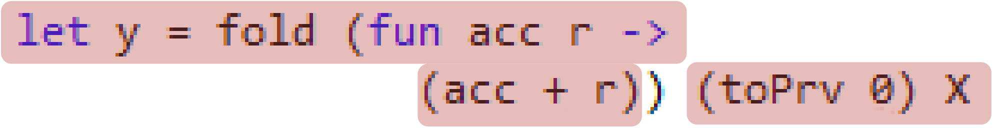
Implemented in C# and F#

- 9995 LoC
- Uses CCI for processing and analysis, operates on IL
- Uses Solver Foundation to resolve constraints

Still a work in progress

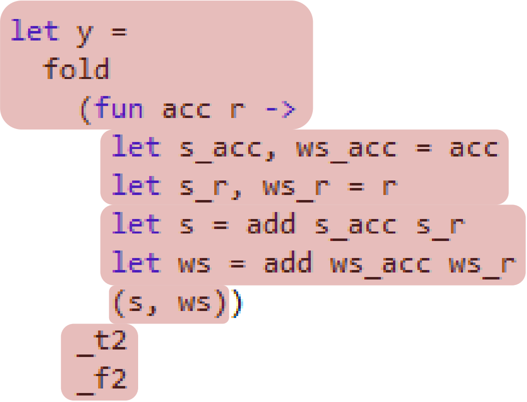
- Integrate cost model generator
- Tune cost model primitive coefficients

Translation in Action



```
let y = fold (fun acc r ->
                          (acc + r)) (toPrv 0) X
```

ZQL



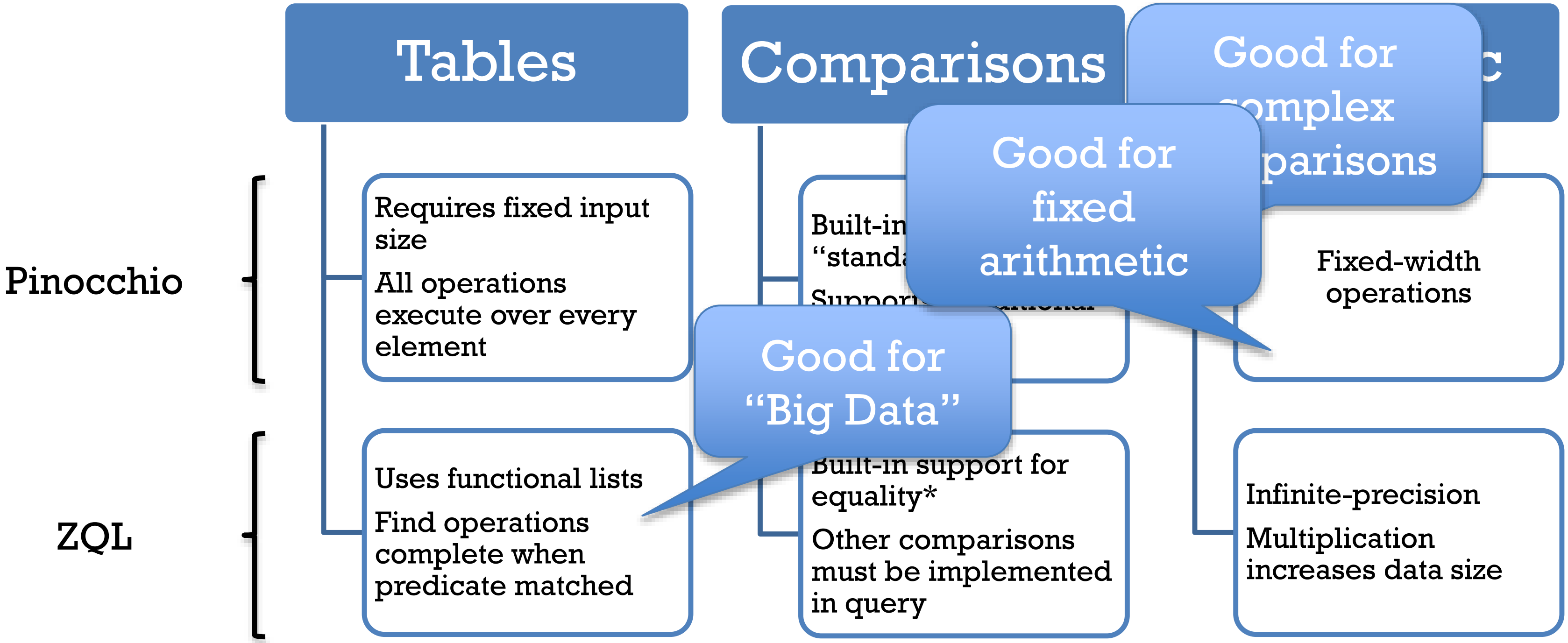
```
let y =
  fold
    (fun acc r ->
      let s_acc, ws_acc = acc
      let s_r, ws_r = r
      let s = add s_acc s_r
      let ws = add ws_acc ws_r
      (s, ws))
    _t2
    _f2
```

Pinocchio

1 Multiplication
100 Additions
101 I/O Wires

```
total 203
input 0 # input
input 1 # input
input 2 # input
input 3 # input
input 4 # input
input 5 # input
input 6 # input
input 7 # input
```

Performance Comparison



	Tables	Comparisons	c
Pinocchio	Requires fixed input size All operations execute over every element	Built-in “standa Support ational	Fixed-width operations
ZQL	Uses functional lists Find operations complete when predicate matched	Built-in support for equality* Other comparisons must be implemented in query	Infinite-precision Multiplication increases data size

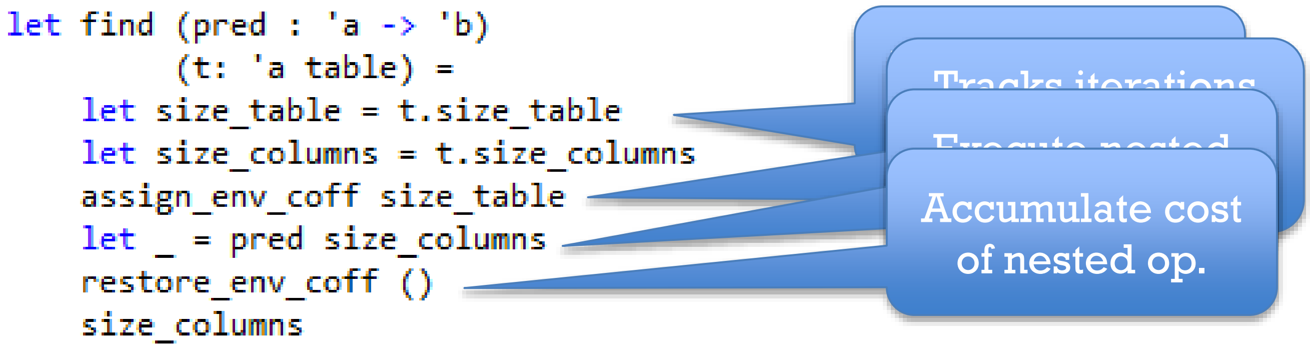
Good for complex parisons

Good for fixed arithmetic

Good for “Big Data”

ZQL Performance

Symbolically execute code generated by ZQL compiler

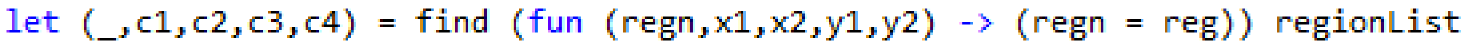


```
let find (pred : 'a -> 'b)
         (t: 'a table) =
    let size_table = t.size_table
    let size_columns = t.size_columns
    assign_env_coff size_table
    let _ = pred size_columns
    restore_env_coff ()
    size_columns
```

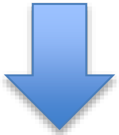
Tracks iterations

Execute nested

Accumulate cost of nested op.



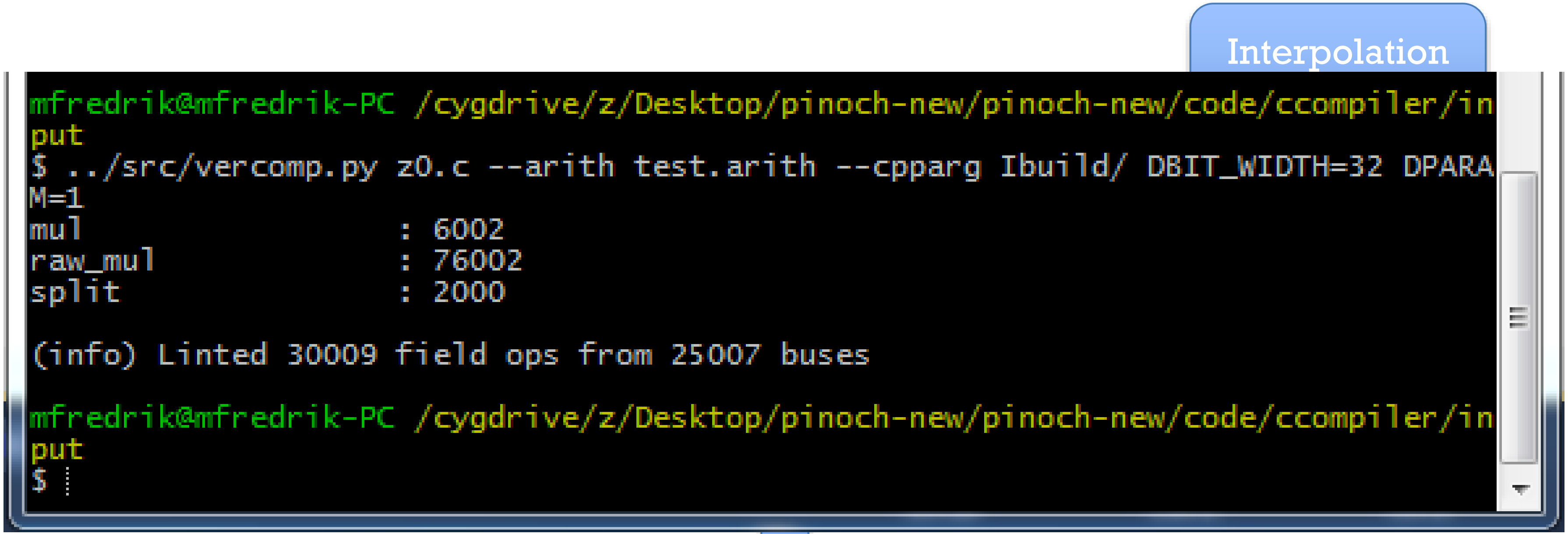
```
let (_,c1,c2,c3,c4) = find (fun (regn,x1,x2,y1,y2) -> (regn = reg)) regionList
```



$$eqOp*regionListSize + \overbrace{addOP + 12*expOp + 3 * extendOp + 14*mltOp + \ldots}^{\text{Cryptographic Overhead}}$$

Pinocchio Performance

Static polynomial based on circuit characteristics



```
mfredrik@mfredrik-PC /cygdrive/z/Desktop/pinoch-new/pinoch-new/code/ccompiler/in
put
$ ../src/vercomp.py z0.c --arith test.arith --cpparg Ibuild/ DBIT_WIDTH=32 DPARA
M=1
mul                     : 6002
raw_mul                 : 76002
split                   : 2000

(info) Linted 30009 field ops from 25007 buses

mfredrik@mfredrik-PC /cygdrive/z/Desktop/pinoch-new/pinoch-new/code/ccompiler/in
put
$
```

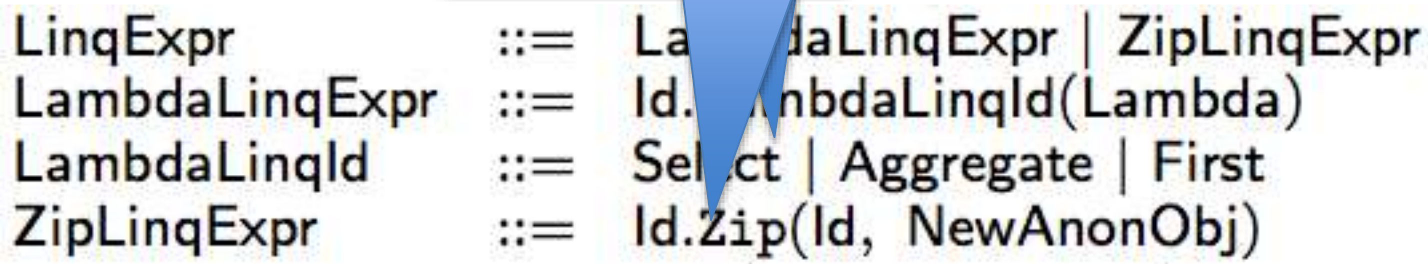
$O(6002 \log^2 6002)(add+mul) + 6507\ ExpT + 44034\ ExpB + 50541\ ExpMulB + \ldots$

Compiling to Zero-Knowledge

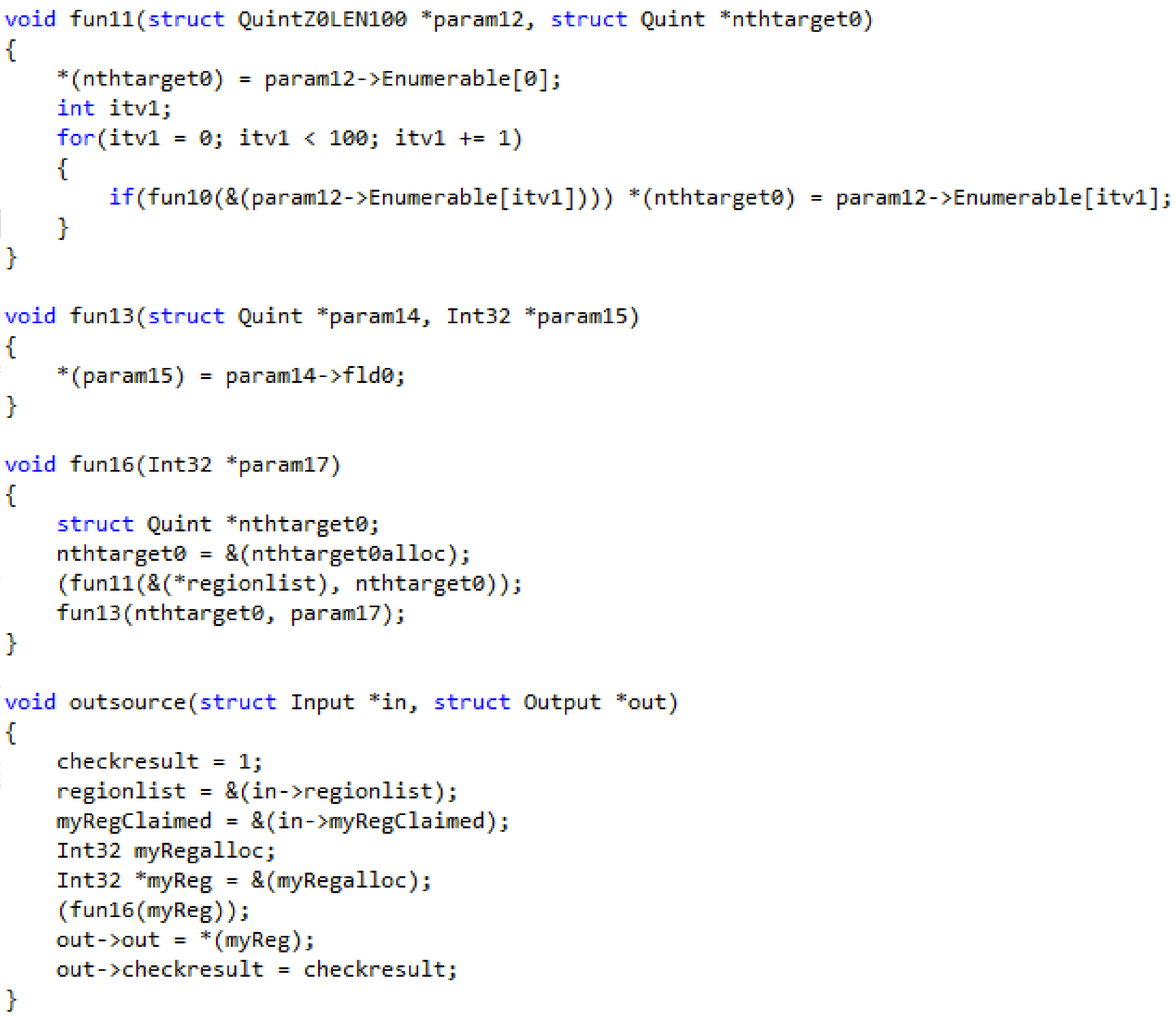
IL
Code

INQ
ions

Combinations of list-structured data



```
LinqExpr         ::=  La    daLinqExpr | ZipLinqExpr
LambdaLinqExpr   ::=  Id.   nbdaLinqId(Lambda)
LambdaLinqId     ::=  Sel ct | Aggregate | First
ZipLinqExpr      ::=  Id.Zip(Id, NewAnonObj)
```



```
void fun11(struct QuintZ0LEN100 *param12, struct Quint *nthtarget0)
{
    *(nthtarget0) = param12->Enumerable[0];
    int itv1;
    for(itv1 = 0; itv1 < 100; itv1 += 1)
    {
        if(fun10(&(param12->Enumerable[itv1]))) *(nthtarget0) = param12->Enumerable[itv1];
    }
}

void fun13(struct Quint *param14, Int32 *param15)
{
    *(param15) = param14->fld0;
}

void fun16(Int32 *param17)
{
    struct Quint *nthtarget0;
    nthtarget0 = &(nthtarget0alloc);
    (fun11(&(*regionlist), nthtarget0));
    fun13(nthtarget0, param17);
}

void outsource(struct Input *in, struct Output *out)
{
    checkresult = 1;
    regionlist = &(in->regionlist);
    myRegClaimed = &(in->myRegClaimed);
    Int32 myRegalloc;
    Int32 *myReg = &(myRegalloc);
    (fun16(myReg));
    out->out = *(myReg);
    out->checkresult = checkresult;
}
```

1. Inf

$con(expr) =$ $\cup \{\varphi\}$

$\{id.elt\}$

$\{id_1, id_2$ $= 1$

$\{id\}$ $d \Rightarrow \Gamma \cup \{\varphi\}$

$\{id\}$

$\{id.n\}$ $id^f_n \geq id_n\}$

$con(id) = \{$ $i = v\}$

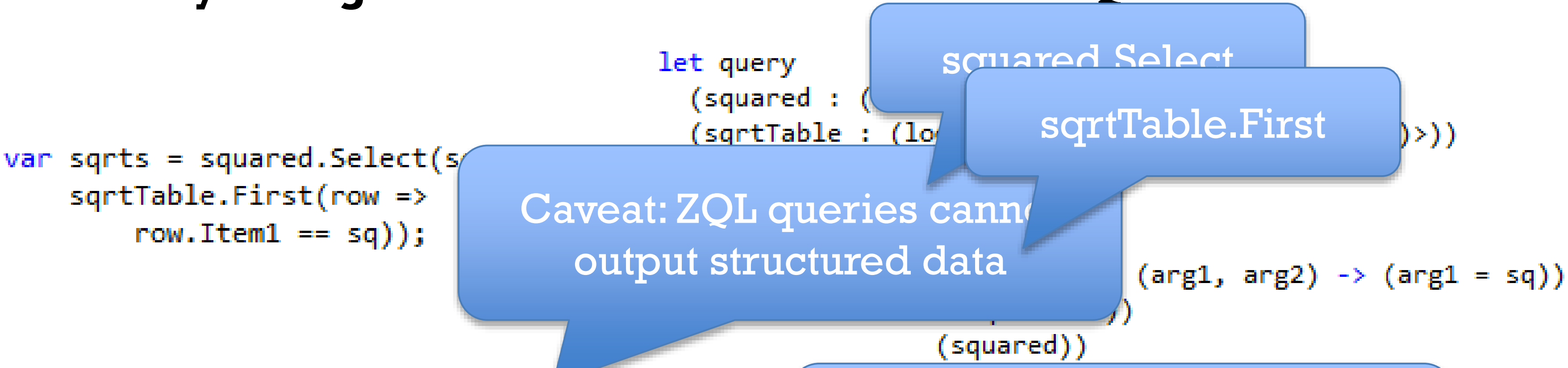
C-Basic Γ, id_1 $= id_3\}$

C-Zip Γ, i $= v\}$

3. En

LINQ -> ZQL

1. **Mostly straightforward translation from LINQ to F#**



```
var sqrts = squared.Select(s
    sqrtTable.First(row =>
        row.Item1 == sq));
```

```
let query
  (squared : (
  (sqrtTable : (lo                                 )>))
                                        (arg1, arg2) -> (arg1 = sq))
                                   ))
                              (squared))
```

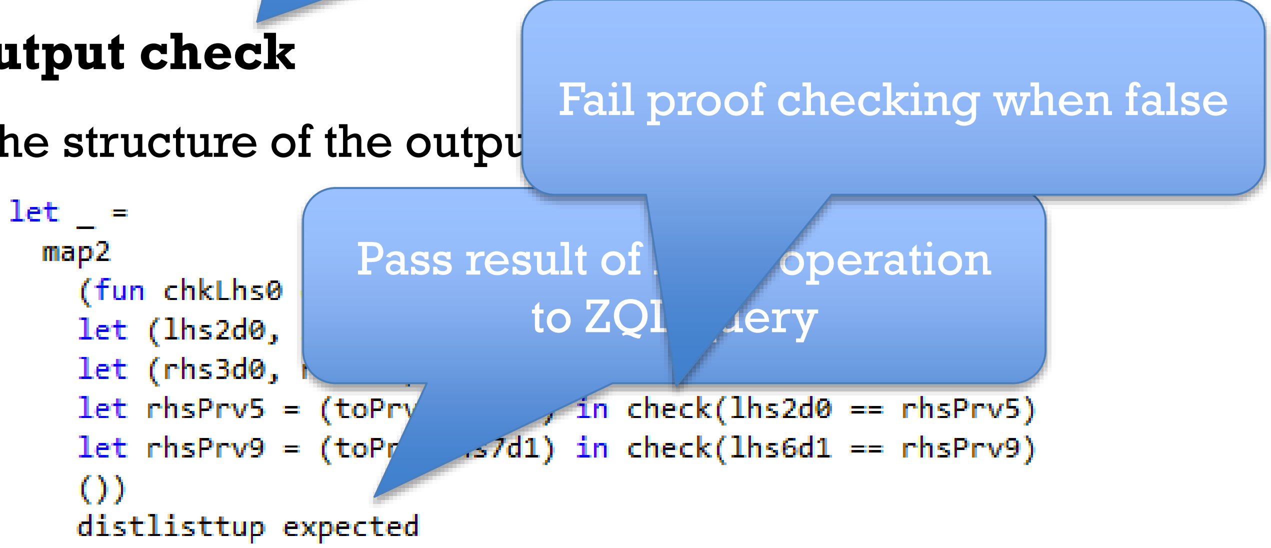
squared.Select

sqrtTable.First

Caveat: ZQL queries canno
output structured data

2. **Generate output check**

Descend on the structure of the outpu



```
let _ =
  map2
    (fun chkLhs0
    let (lhs2d0,
    let (rhs3d0,
    let rhsPrv5 = (toPr         ) in check(lhs2d0 == rhsPrv5)
    let rhsPrv9 = (toPr    s7d1) in check(lhs6d1 == rhsPrv9)
    ())
    distlisttup expected
```

Fail proof checking when false

Pass result of   operation
to ZQL   uery

< Demo >

Back to our example…

- Z — Look up region in a large table of coordinates
- P — Show that GPS coordinates match result
- P — Encode region as a vector
- ZP — Creates shares of vector
- ZP — Sums other clients' shares

Distributing Across Tiers

Core Principle: Rely on runtime whenever possible

Minimize the role of the compiler:

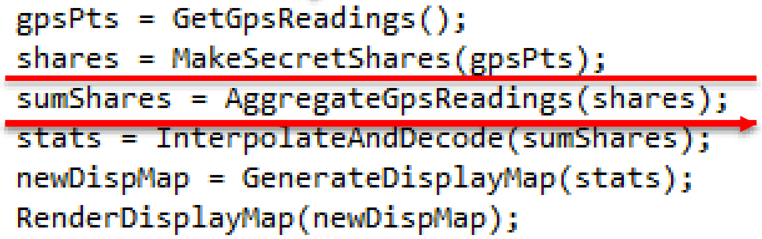
1. een tiers
2. whenever

depe ies exist

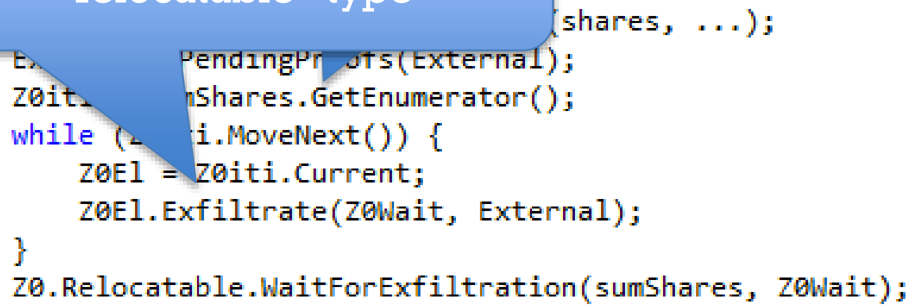
Only the main function can call code on multiple tiers

Functions called from main Enumerable

Each element inherits from “relocatable” type



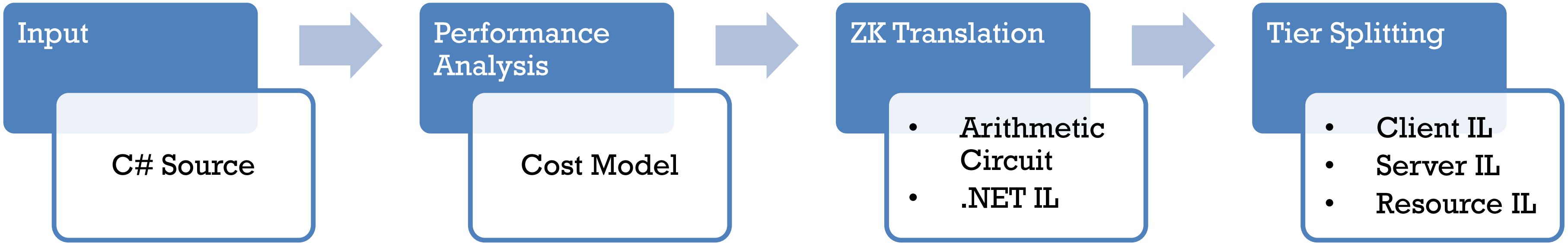
```
gpsPts = GetGpsReadings();
shares = MakeSecretShares(gpsPts);
sumShares = AggregateGpsReadings(shares);
stats = InterpolateAndDecode(sumShares);
newDispMap = GenerateDisplayMap(stats);
RenderDisplayMap(newDispMap);
```



```
                            (shares, ...);
E         PendingPr  ofs(External);
Z0it        mShares.GetEnumerator();
while (     i.MoveNext()) {
    Z0El = Z0iti.Current;
    Z0El.Exfiltrate(Z0Wait, External);
}
Z0.Relocatable.WaitForExfiltration(sumShares, Z0Wait);
```

ZØ: An Optimizing Compiler for ZK

ZØ uses the best of both back-ends as appropriate for the application at hand



Input

C# Source

Performance Analysis

Cost Model

ZK Translation

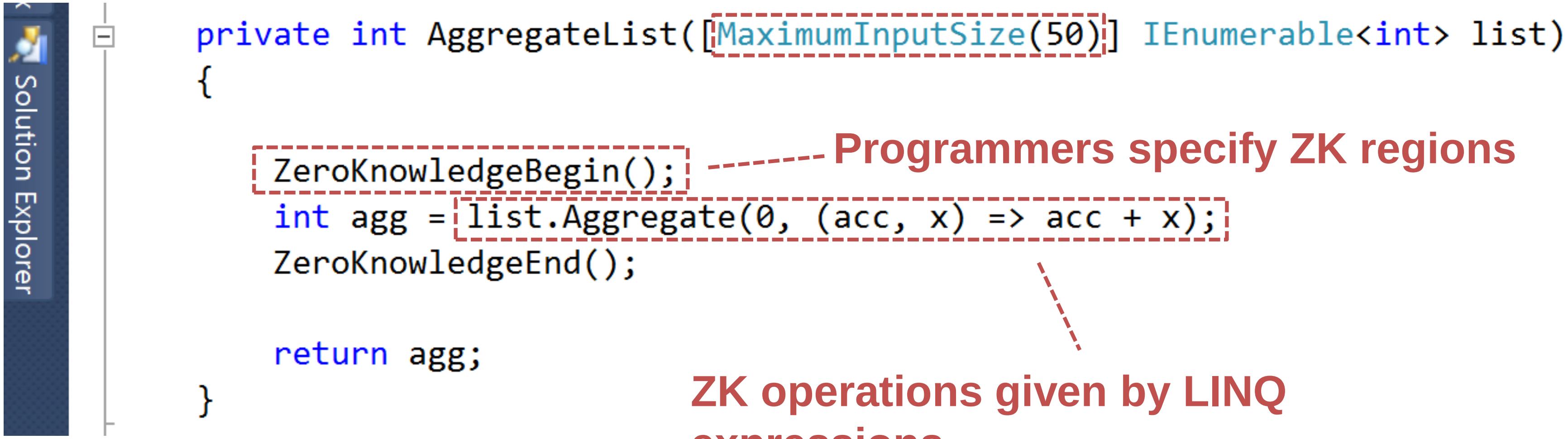
- Arithmetic Circuit
- .NET IL

Tier Splitting

- Client IL
- Server IL
- Resource IL

Translating C# To Zero-Knowledge

Specify ZK input sizes to help optimization



```
private int AggregateList([MaximumInputSize(50)] IEnumerable<int> list)
{

    ZeroKnowledgeBegin();
    int agg = list.Aggregate(0, (acc, x) => acc + x);
    ZeroKnowledgeEnd();

    return agg;
}
```

Programmers specify ZK regions

ZK operations given by LINQ expressions