DIZK

A Distributed Zero Knowledge Proof System

Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, Ion Stoica
University of California Berkeley

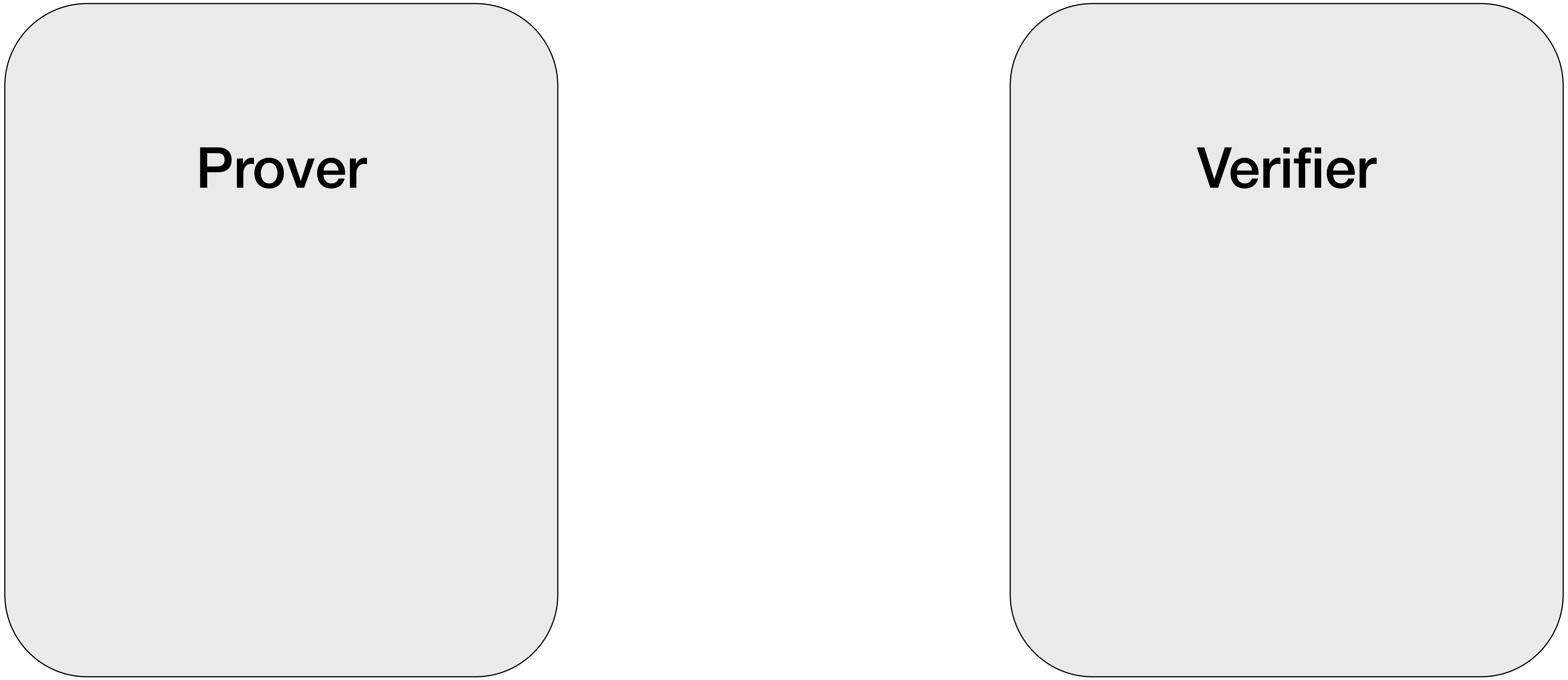
27th USENIX Security Symposium
August 16, 2018

Zero Knowledge Proof

[GMR 85]

Zero Knowledge Proof

[GMR 85]



Zero Knowledge Proof

[GMR 85]

Prover

F	function
y	claimed output

Verifier

F	function
y	claimed output

Zero Knowledge Proof

[GMR 85]

I know x s.t. F(x) = y

Prover

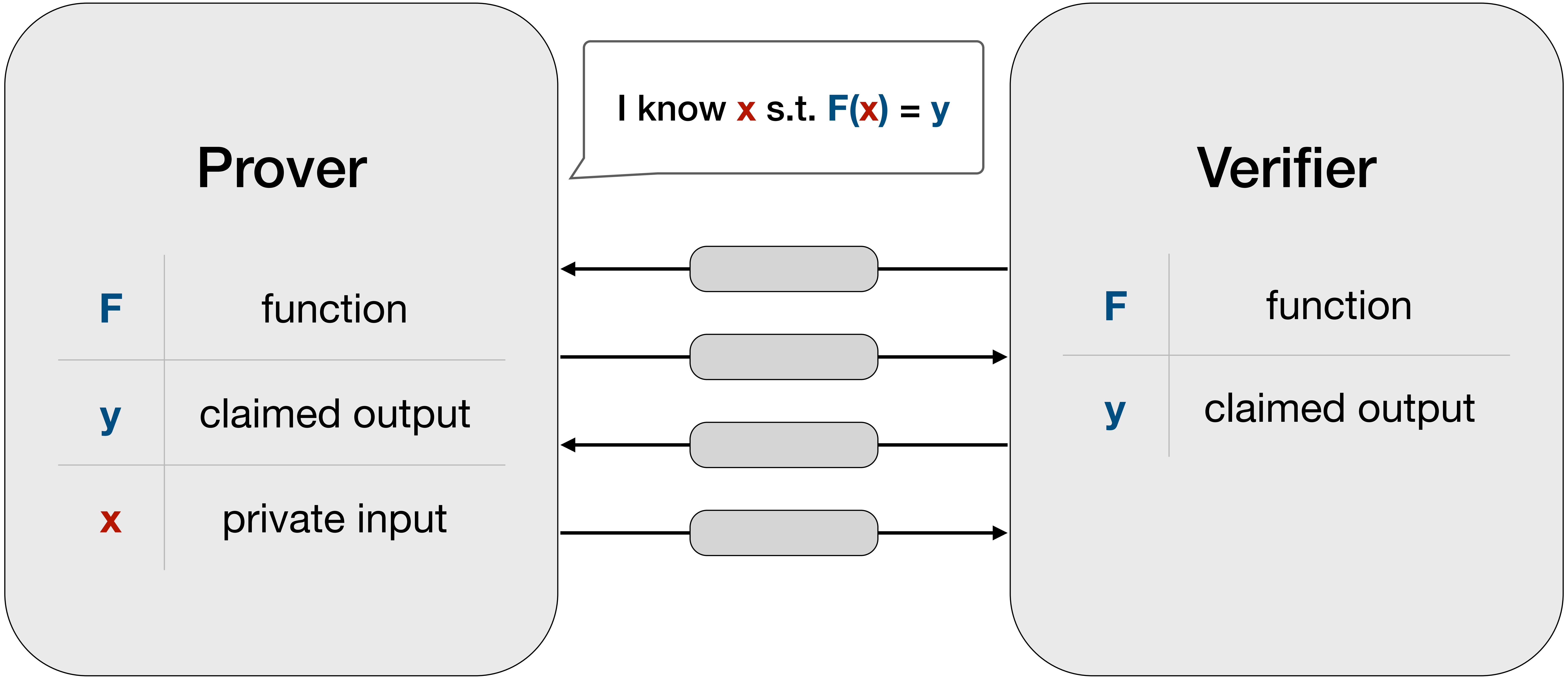
F	function
y	claimed output
x	private input

Verifier

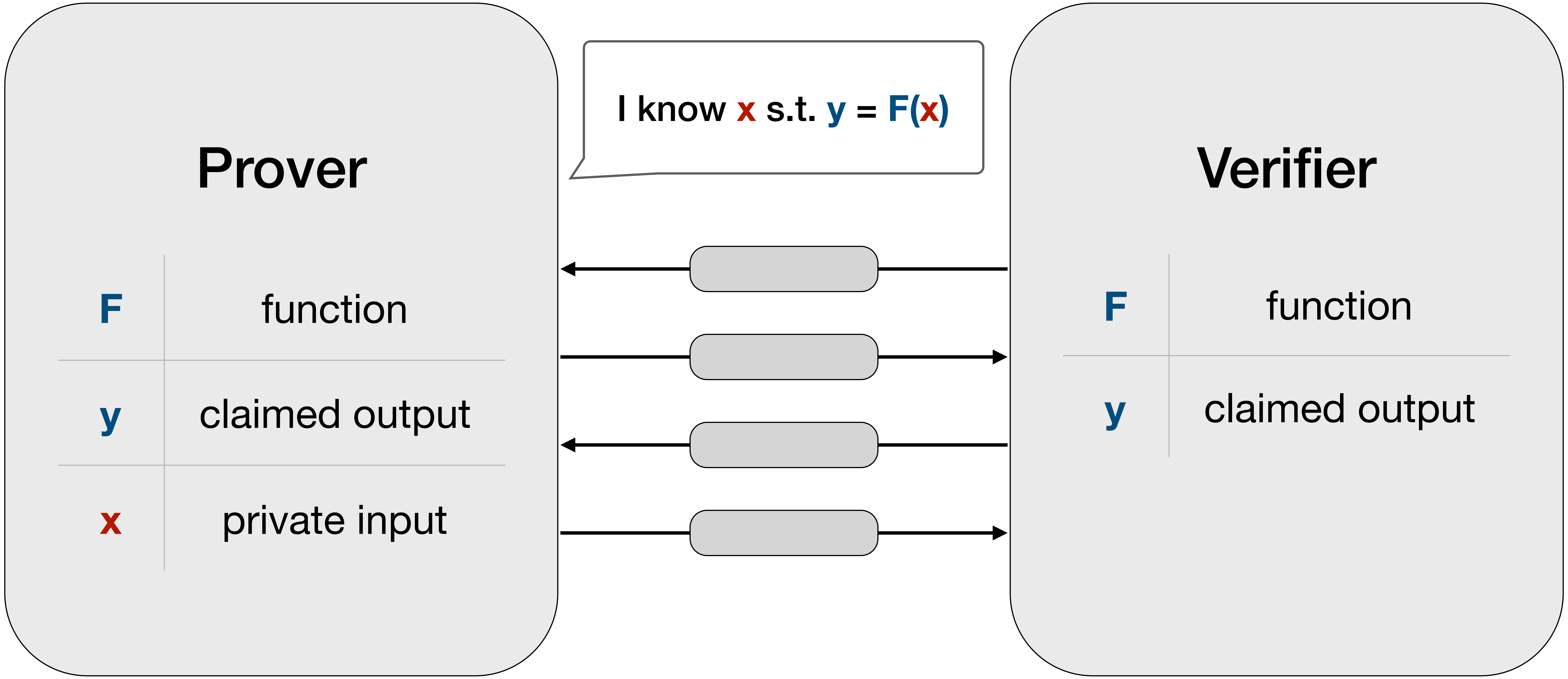
F	function
y	claimed output

Zero Knowledge Proof

[GMR 85]

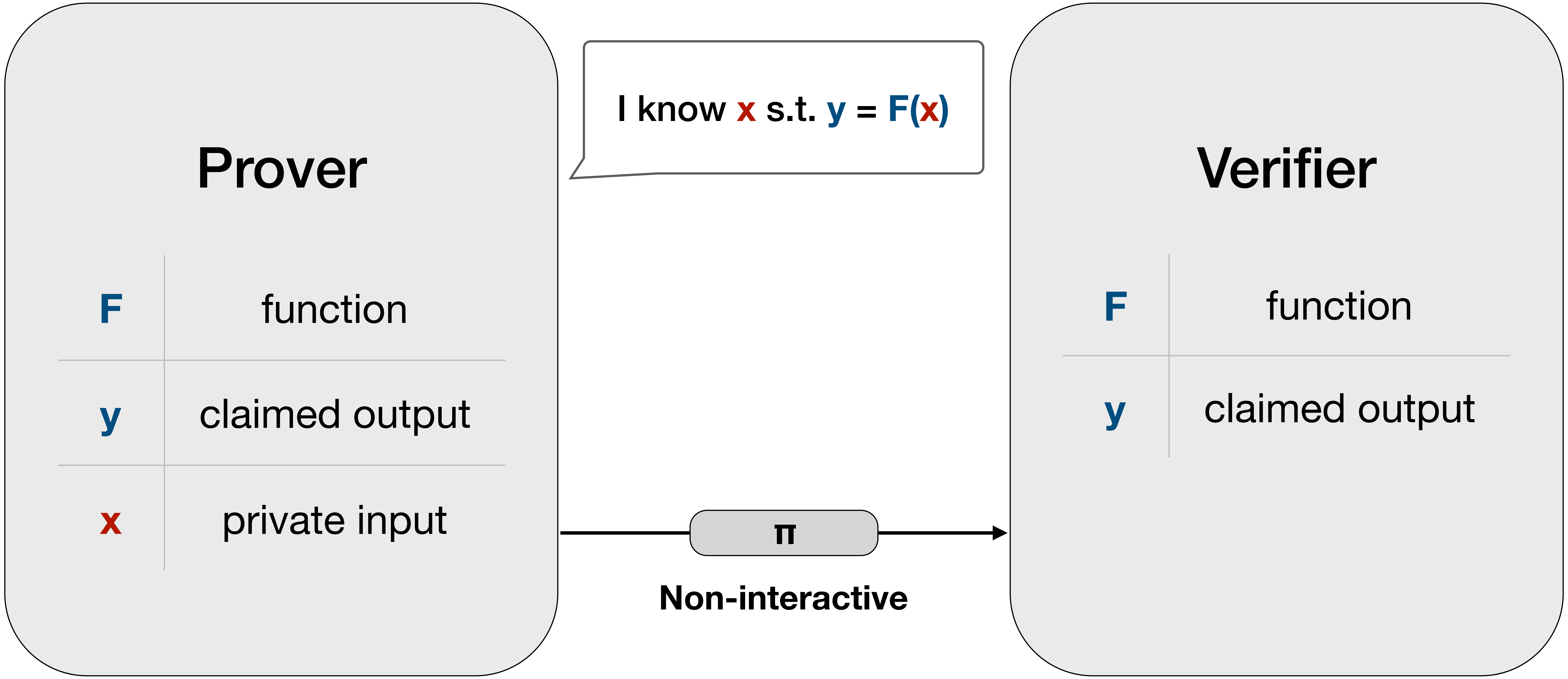


zkSNARK



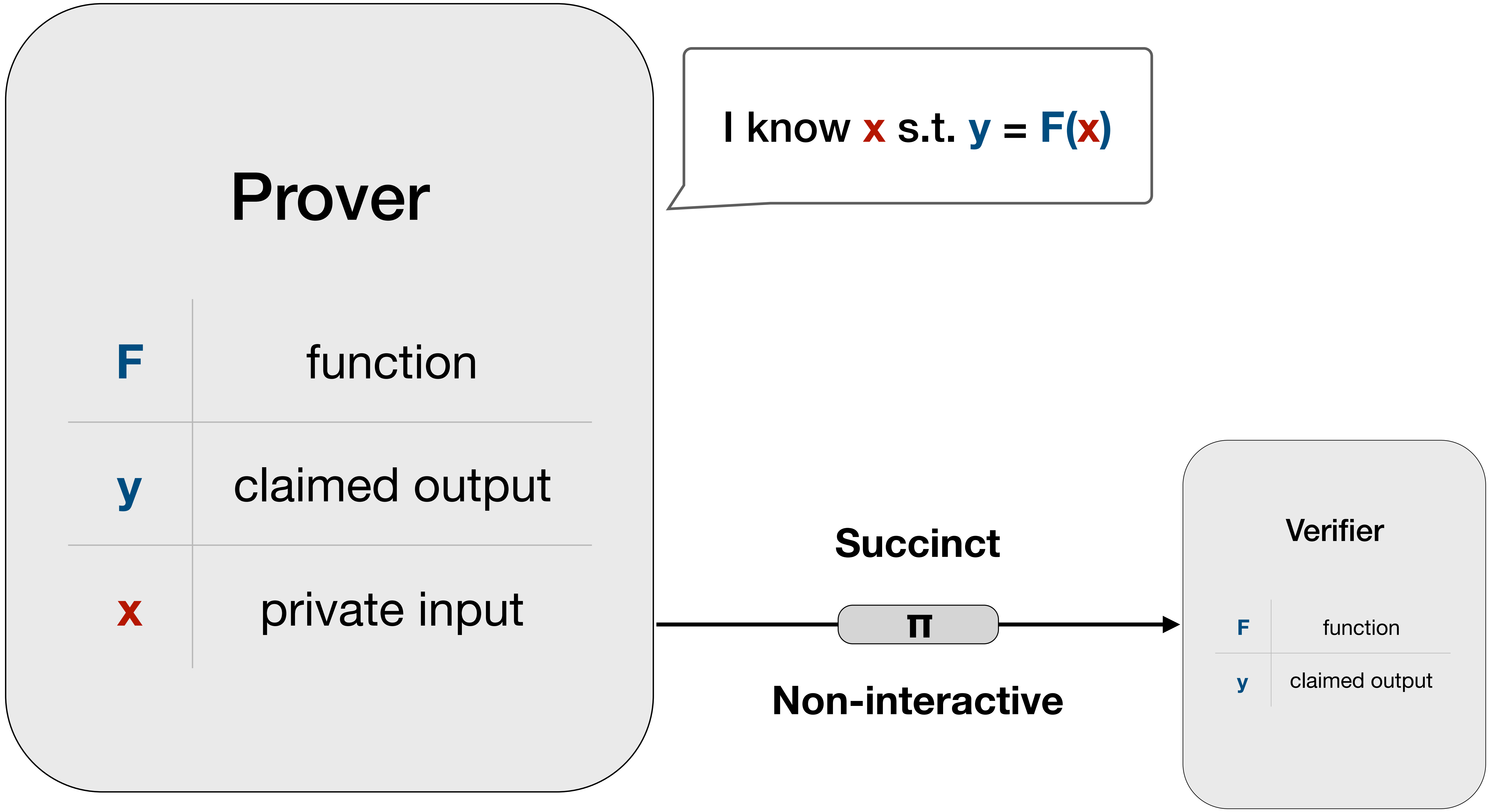
I know x s.t. y = F(x)
Prover
F
function
y
claimed output
x
private input
Verifier
F
function
y
claimed output

zkSNARK

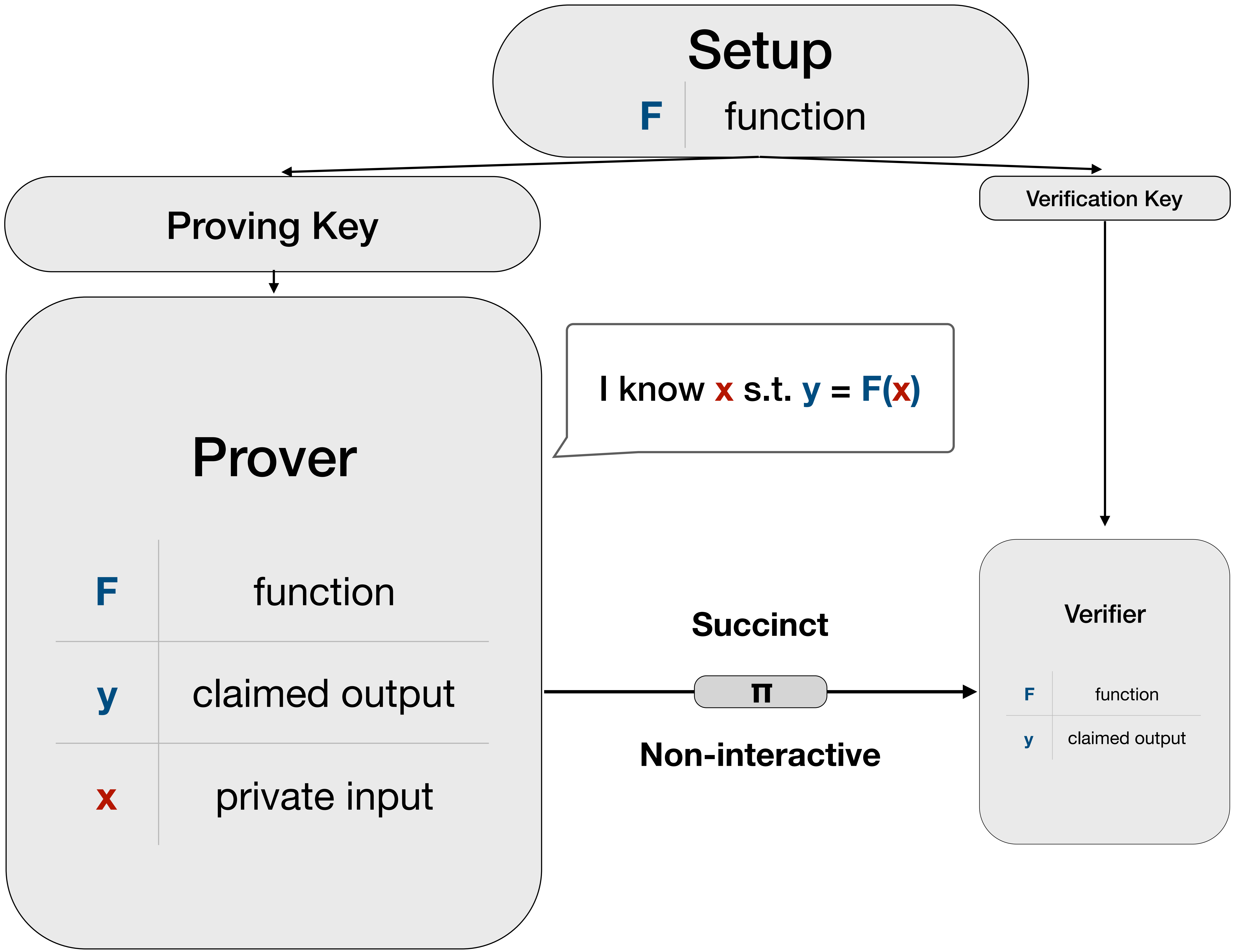


Prover
F function
y claimed output
x private input
I know x s.t. y = F(x)
π
Non-interactive
Verifier
F function
y claimed output

zkSNARK


Prover
F
function
y
claimed output
x
private input
I know x s.t. y = F(x)
Succinct
π
Non-interactive
Verifier
F
function
y
claimed output

preprocessing zkSNARK

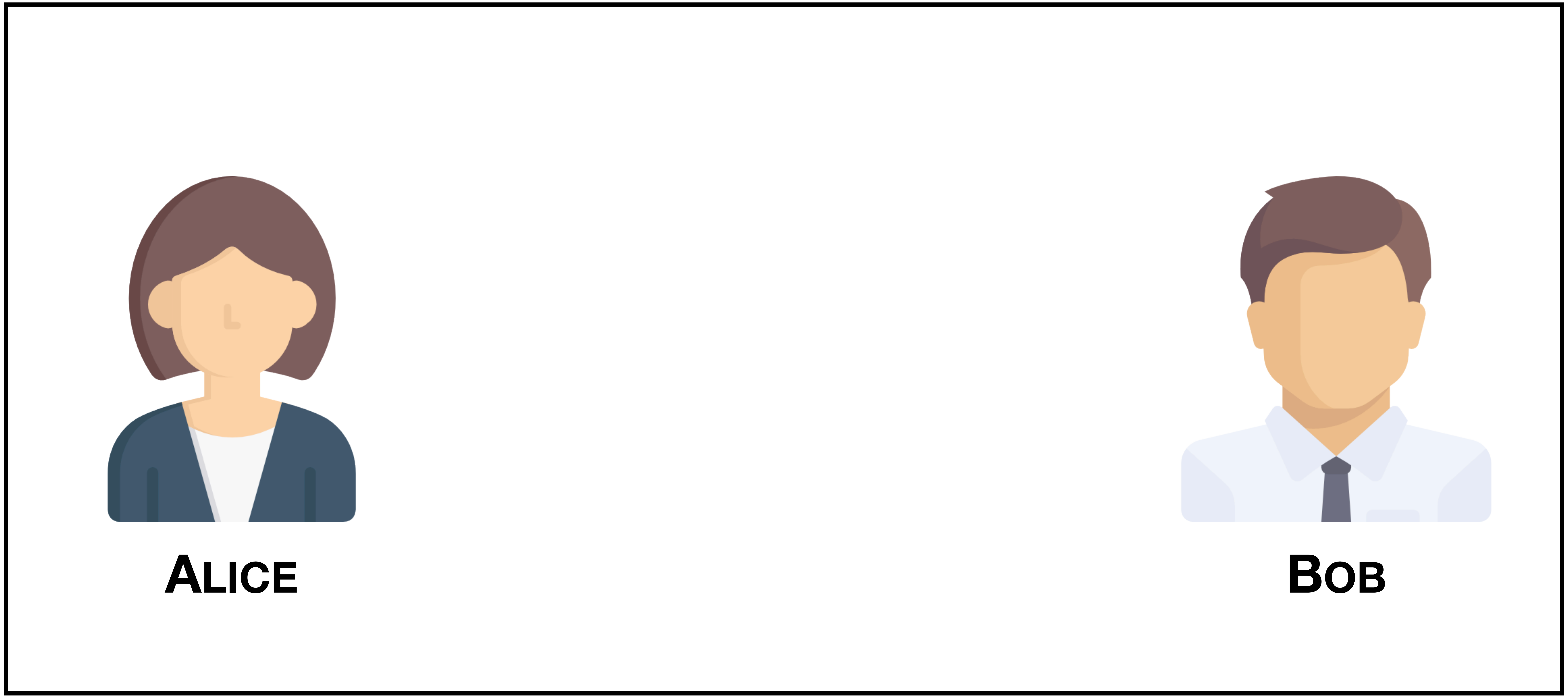


Application #1

[BCGGMTV 14]

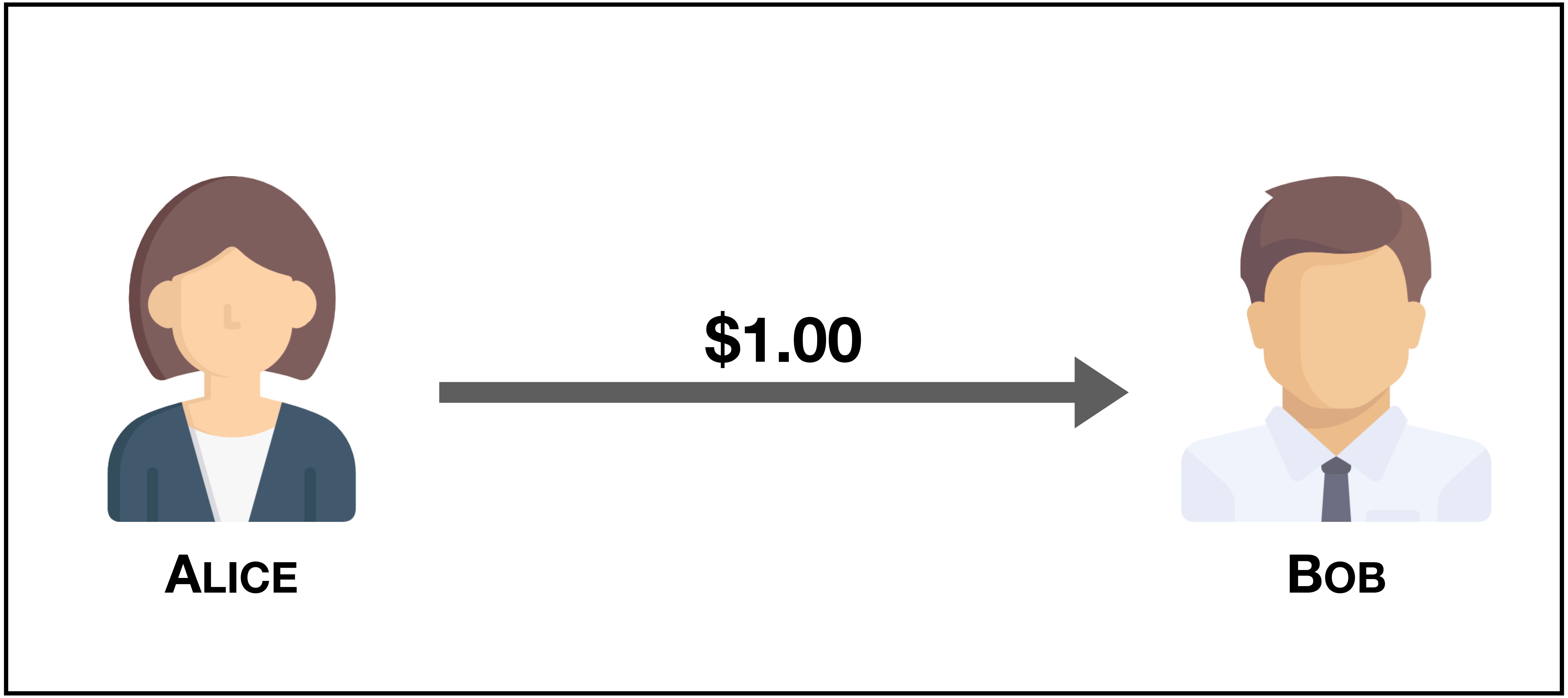
Application #1

[BCGGMTV 14]



Application #1

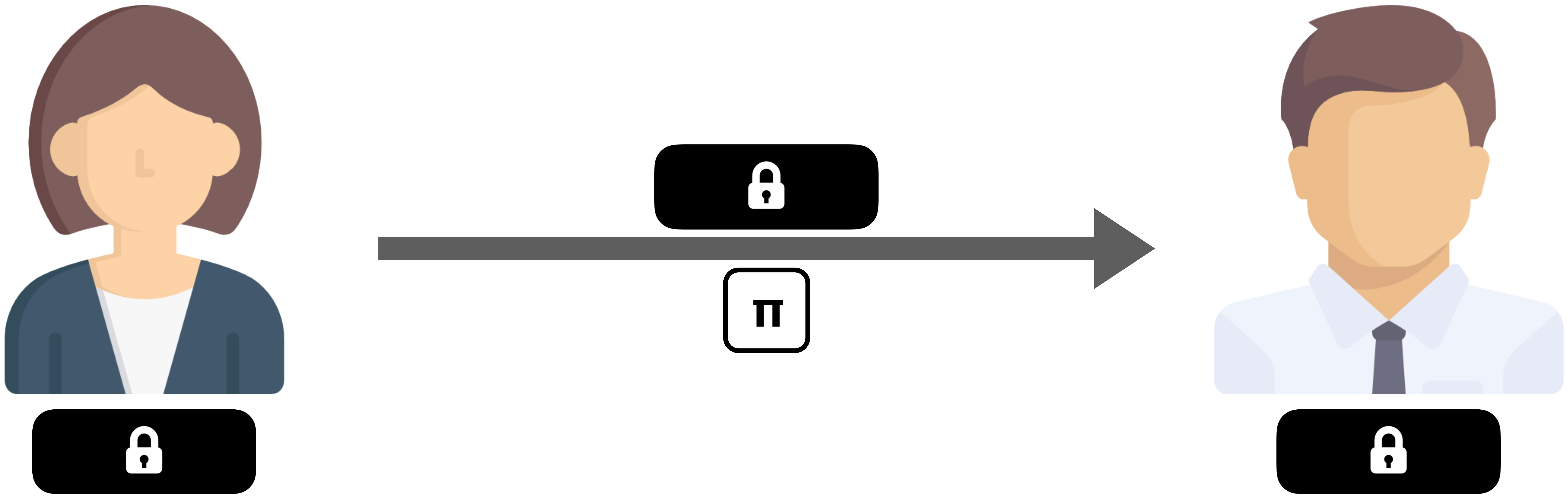
[BCGGMTV 14]



Application #1

[BCGGMTV 14]

Anonymous P2P Payments



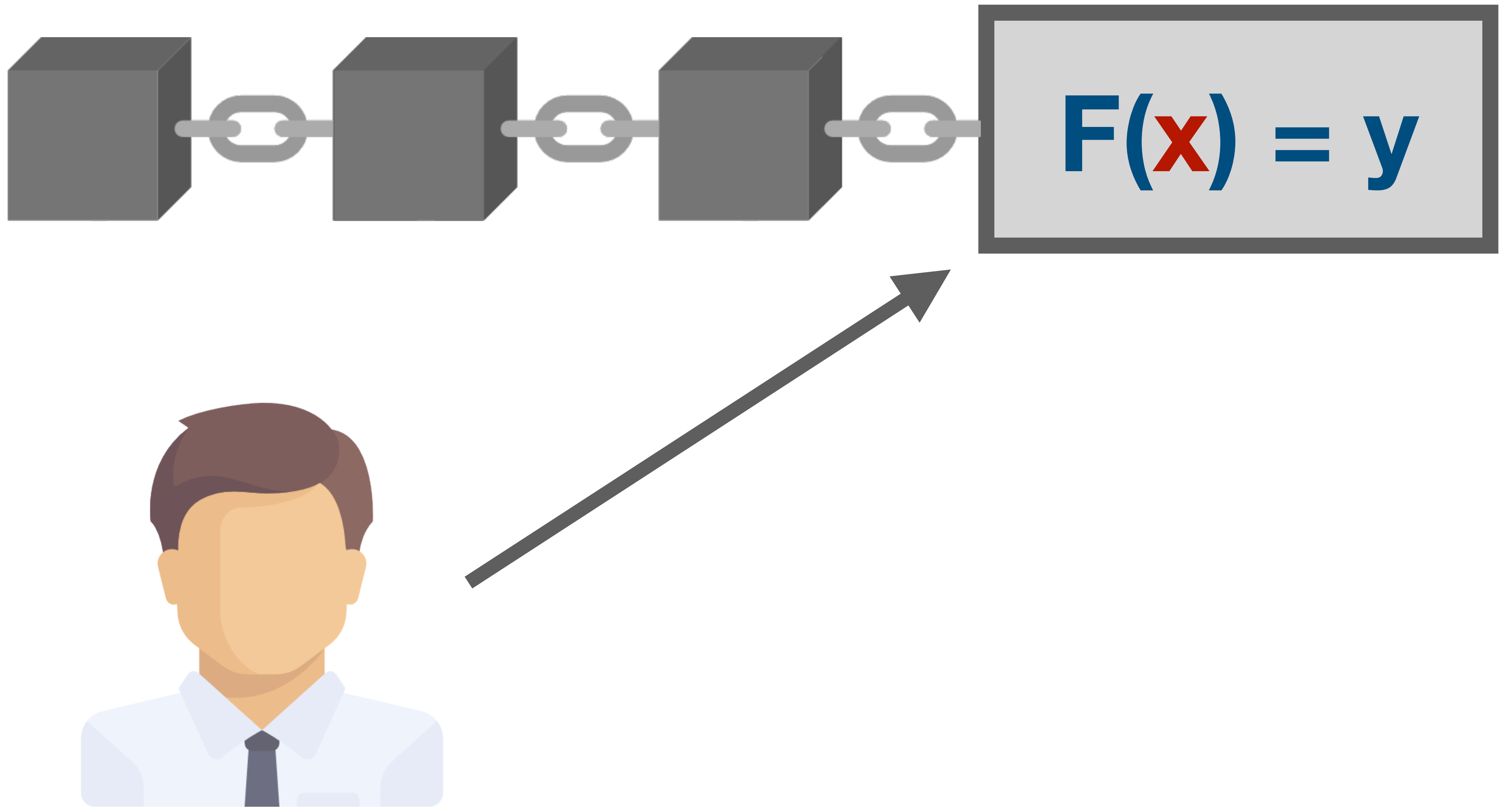
Application #2

Application #2

Smart Contracts

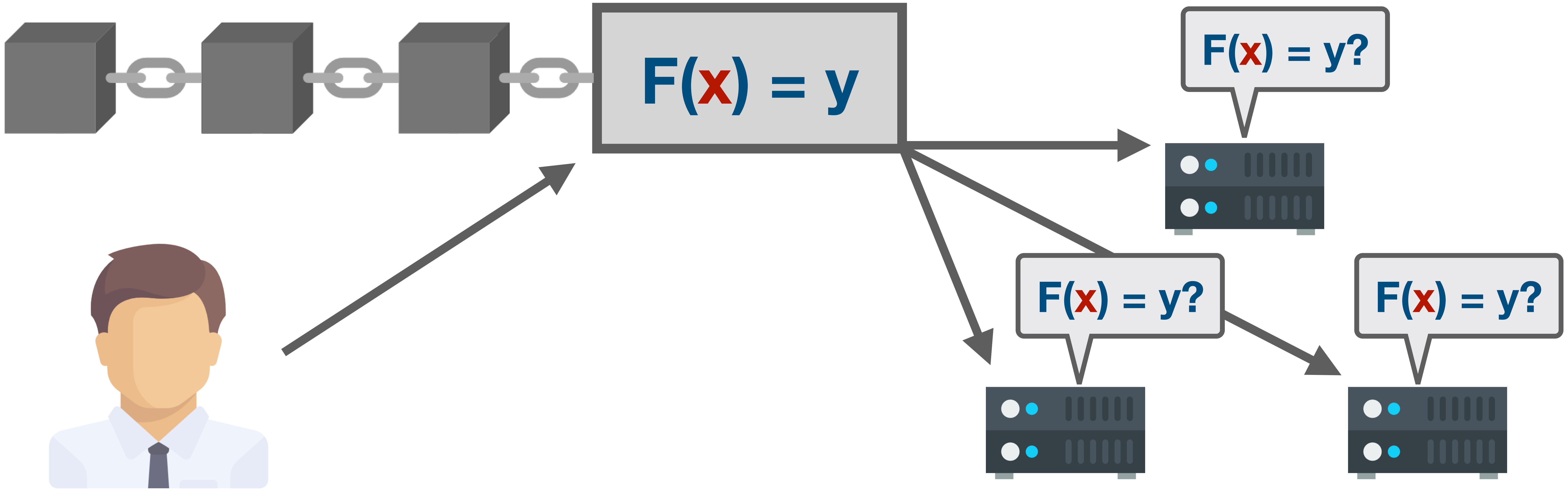
Application #2

Smart Contracts



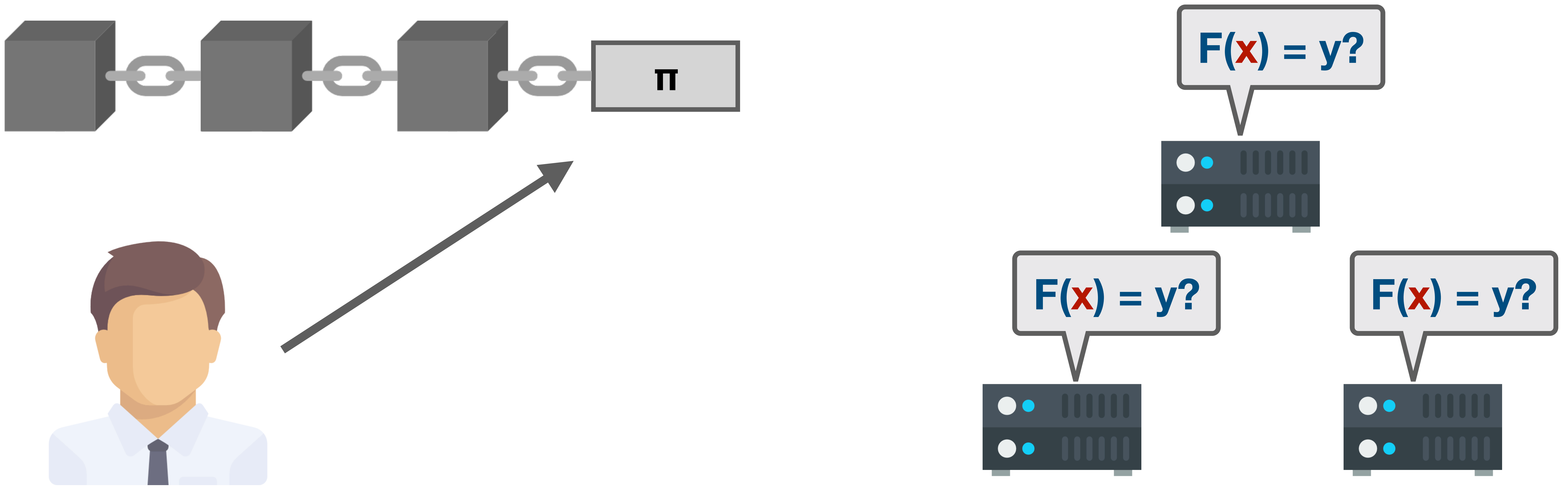
Application #2

Smart Contracts



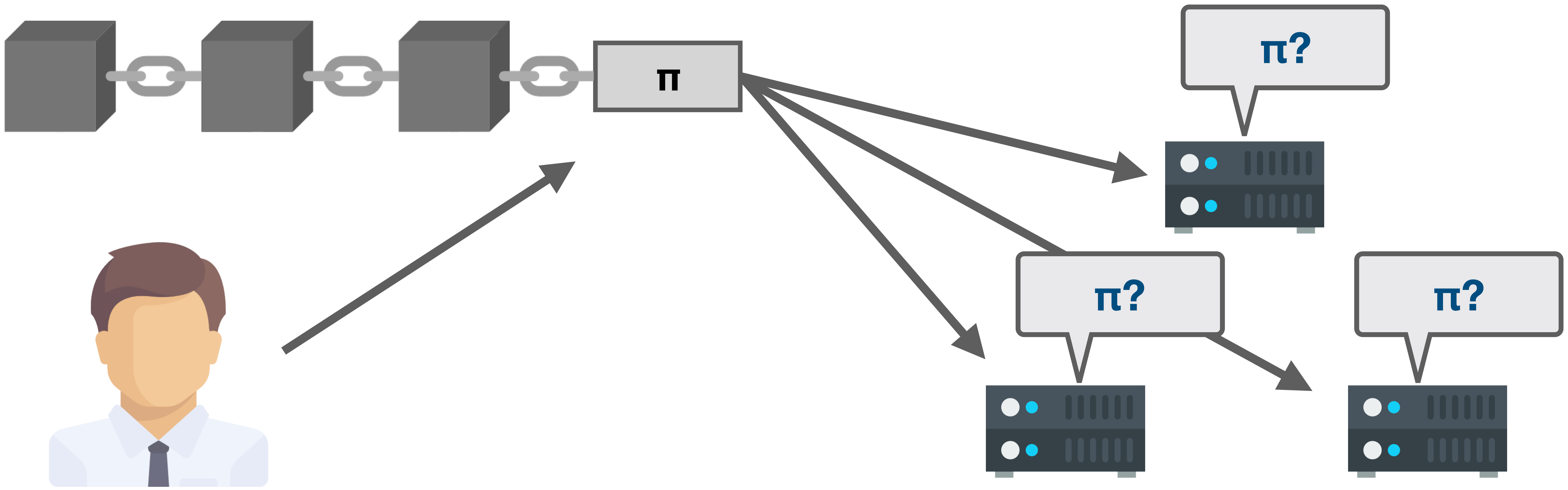
Application #2

Smart Contracts



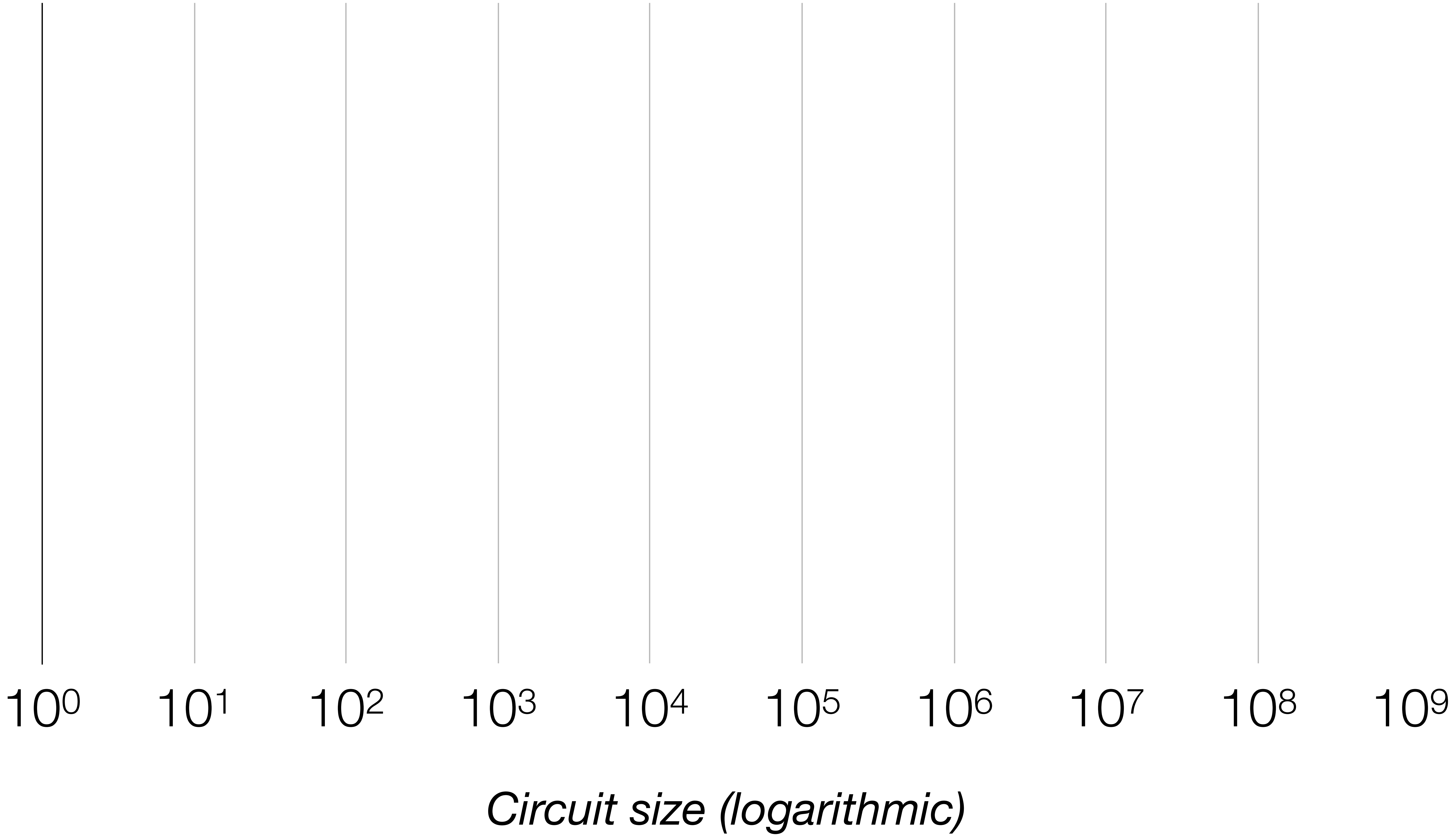
Application #2

Smart Contracts

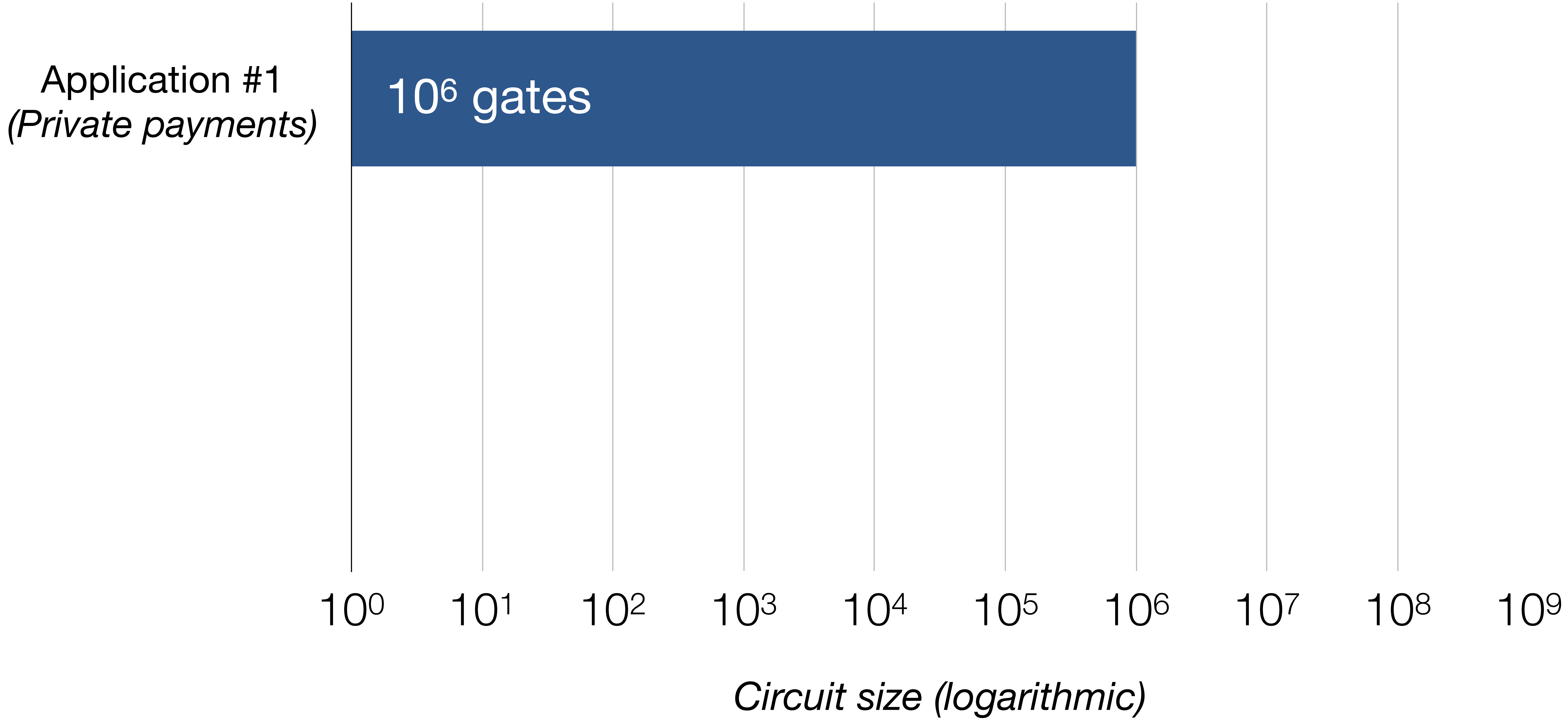


Good News & Bad News

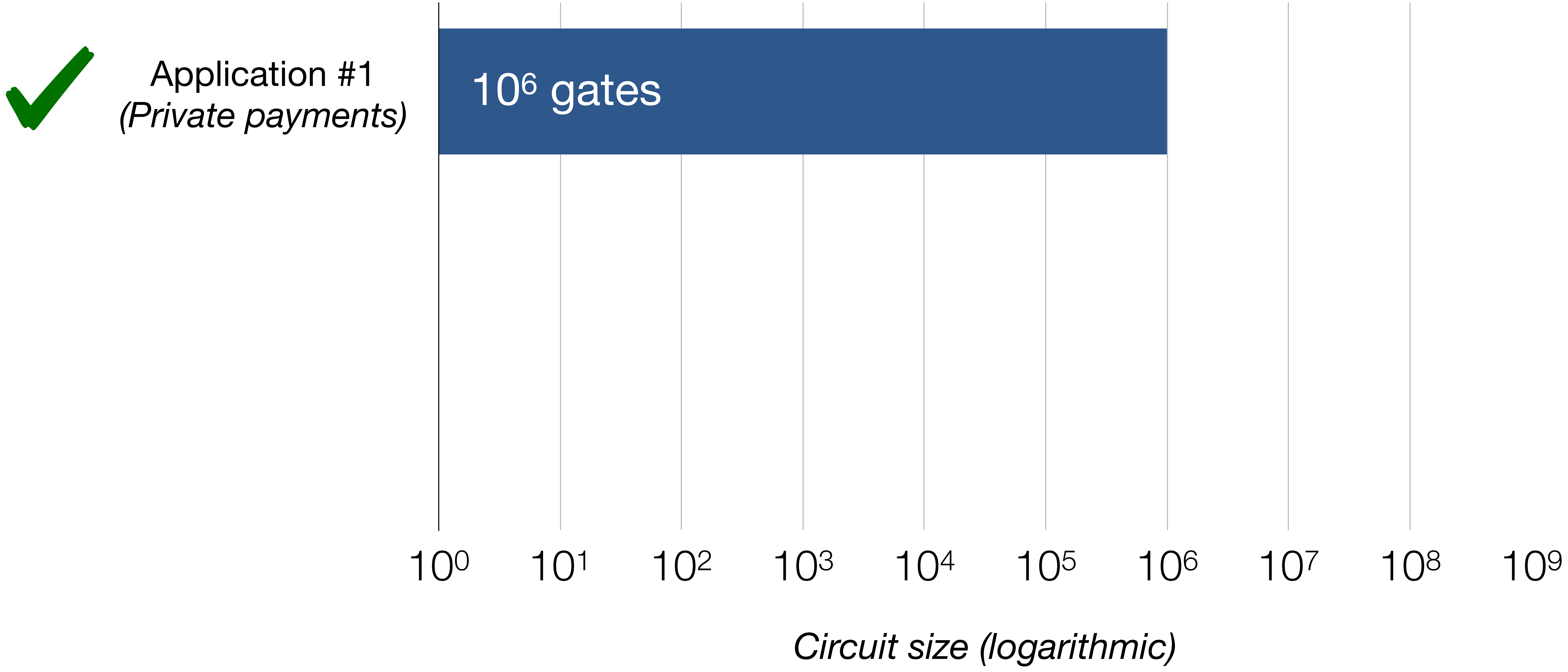
Good News & Bad News



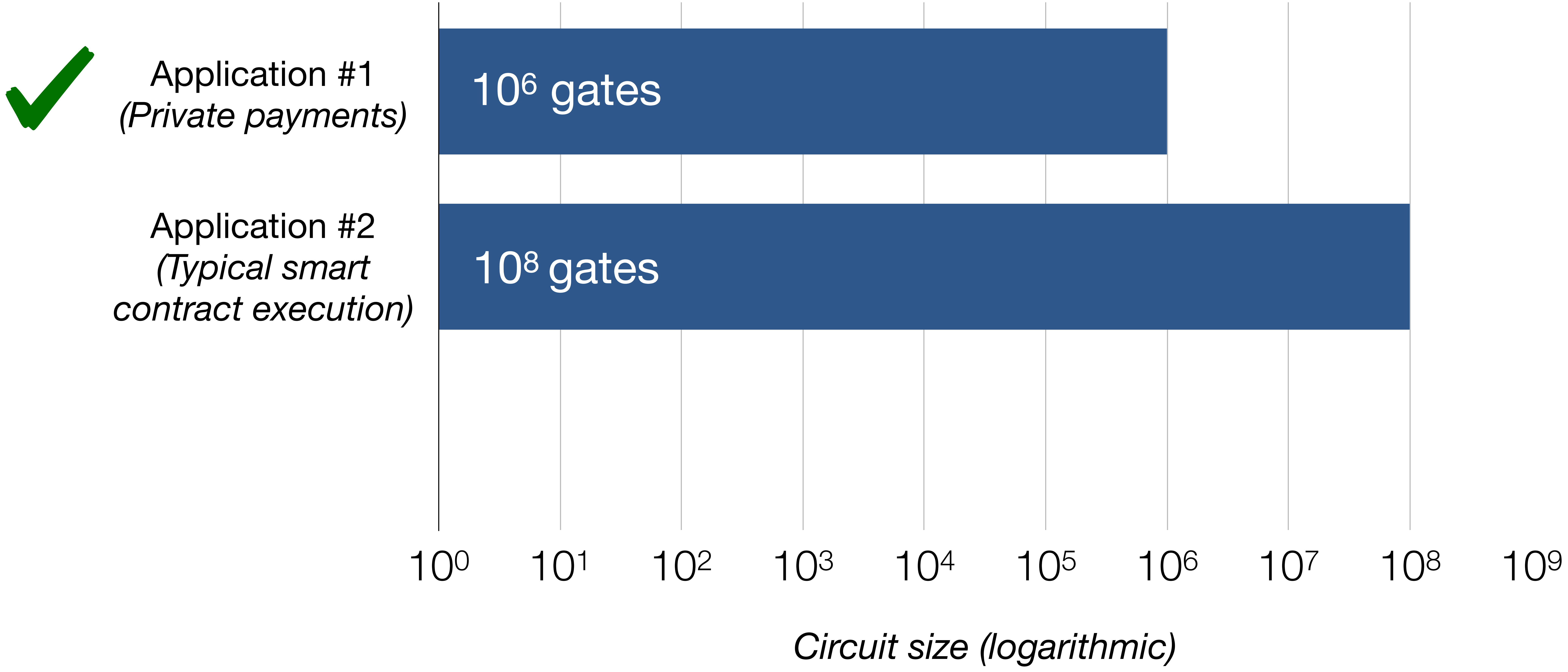
Good News & Bad News



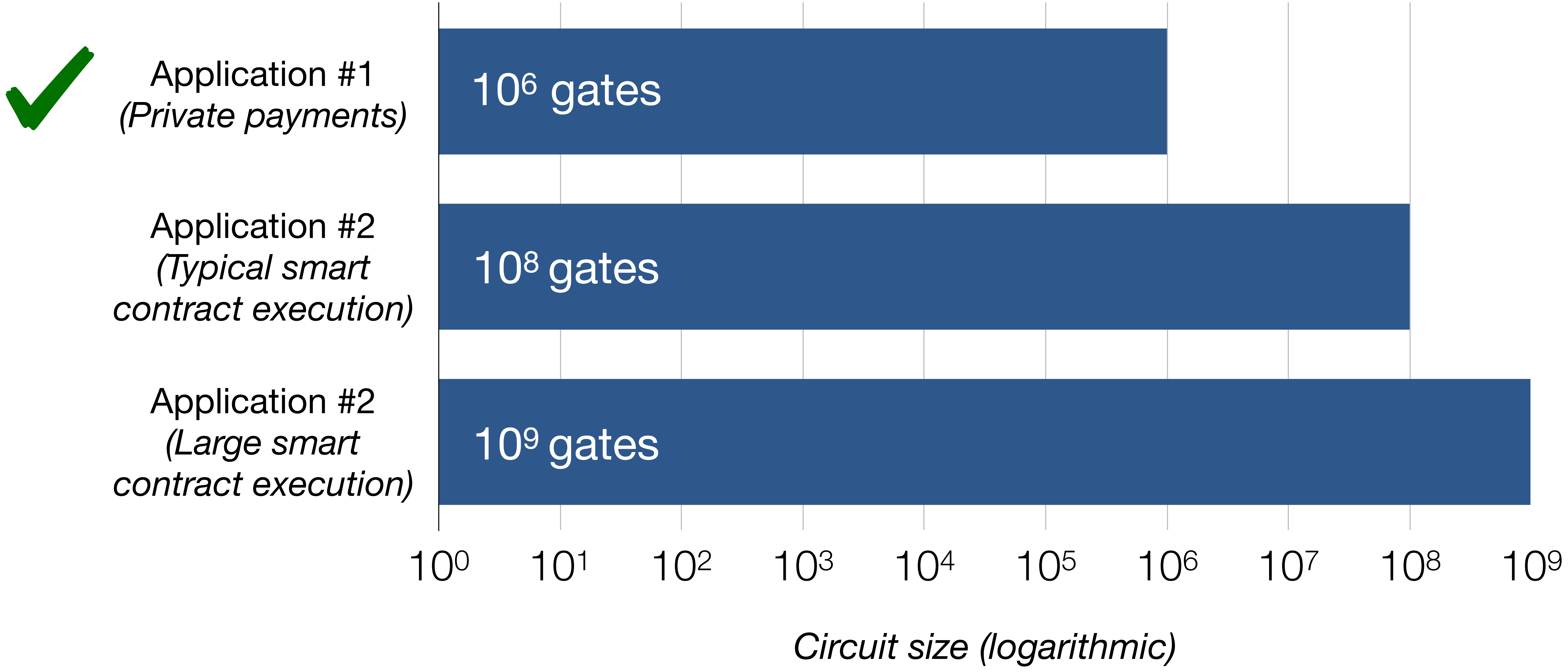
Good News & Bad News



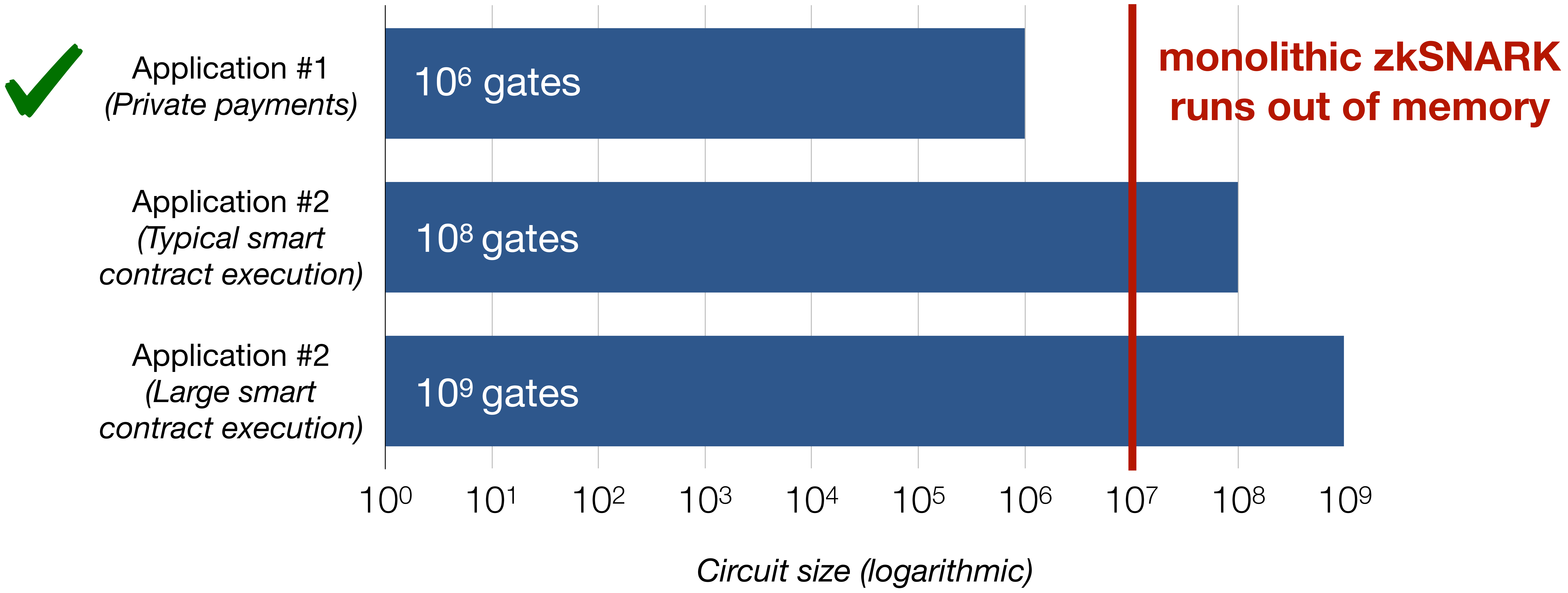
Good News & Bad News



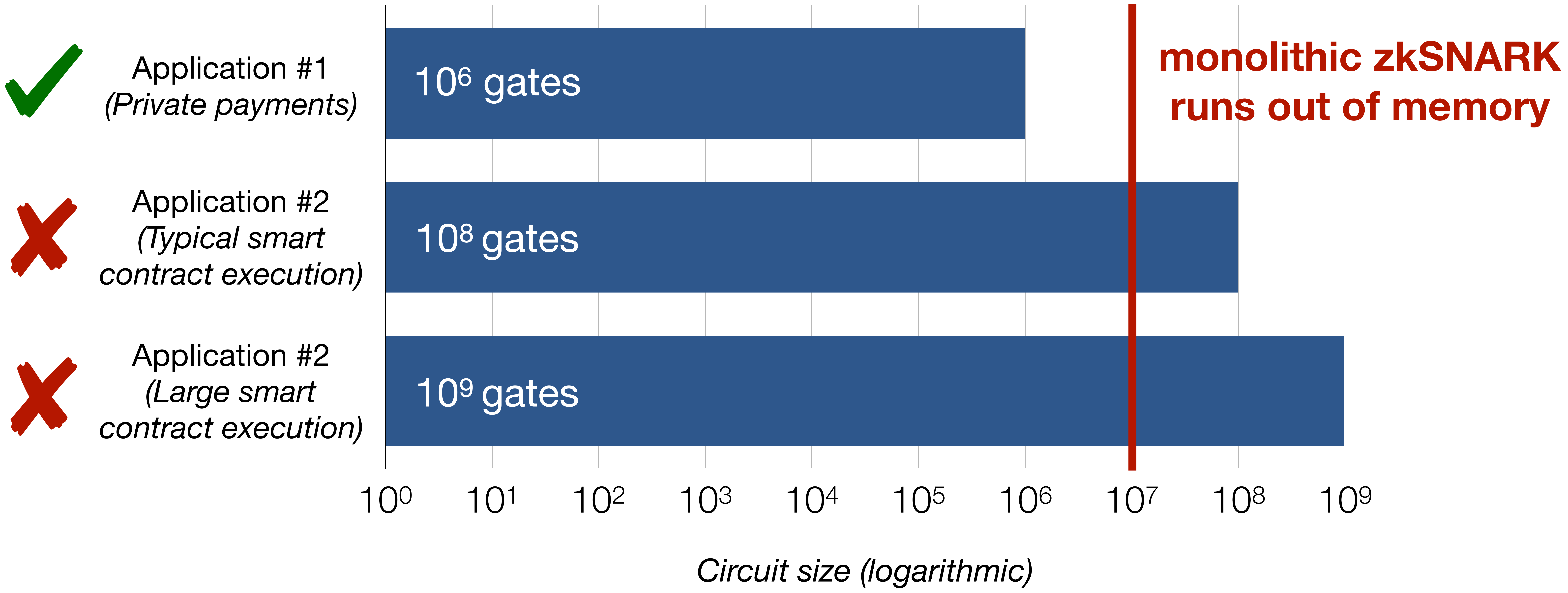
Good News & Bad News



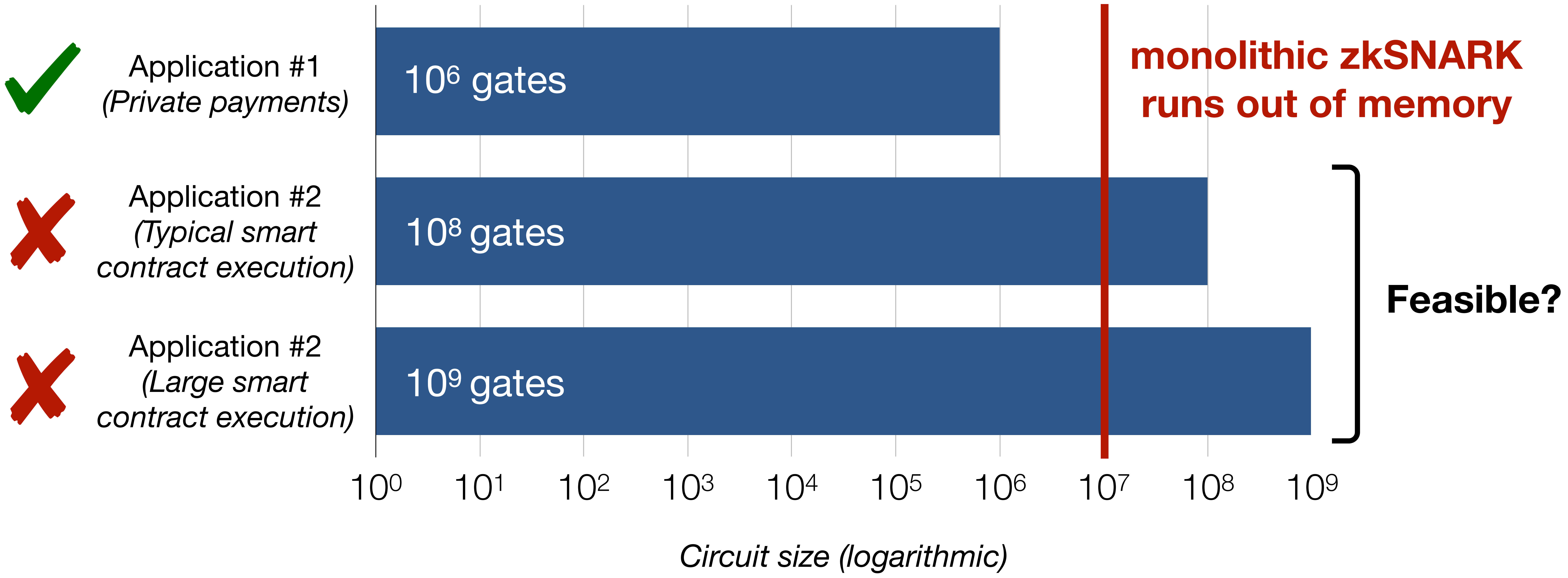
Good News & Bad News



Good News & Bad News



Good News & Bad News



DIZK is a zero knowledge proof system that is:

DIZK is a zero knowledge proof system that is:

DISTRIBUTED

Enables the execution of a zkSNARK
Setup and Prover across a **compute cluster**

DIZK is a zero knowledge proof system that is:

DISTRIBUTED

Enables the execution of a zkSNARK
Setup and Prover across a **compute cluster**

SCALABLE

Reaches heretofore unreachable circuit sizes (up to ***billions*** of gates)

Double the number of machines → twice the circuit size

DIZK is a zero knowledge proof system that is:

Distributed

Enables the execution of a zkSNARK
Setup and Prover across a **compute cluster**

Scalable

Reaches heretofore unreachable circuit sizes (up to ***billions*** of gates)

Double the number of machines → twice the circuit size

Parallel

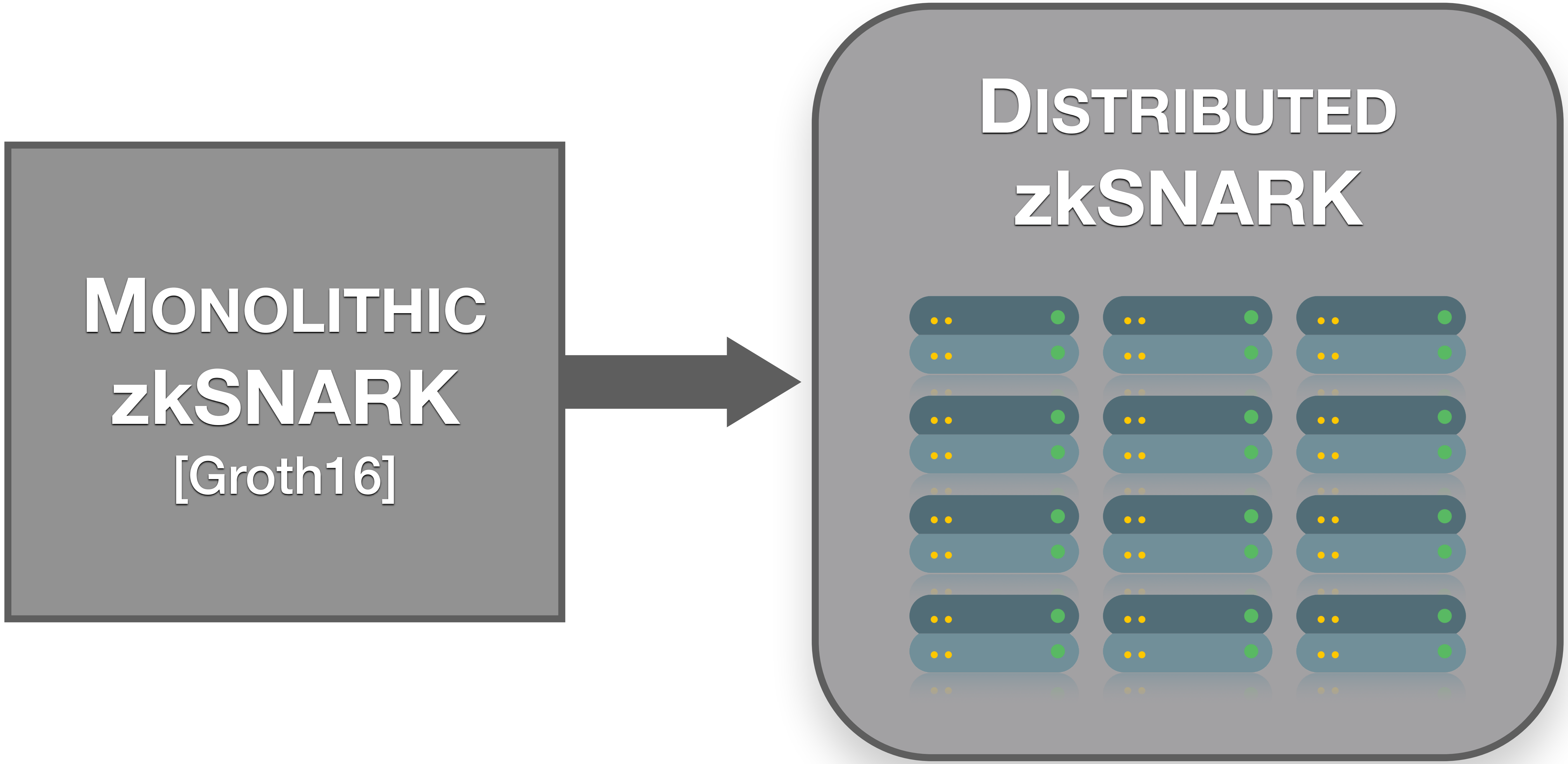
Speeds up the time it takes to generate a proof

Double the number of machines → twice as fast

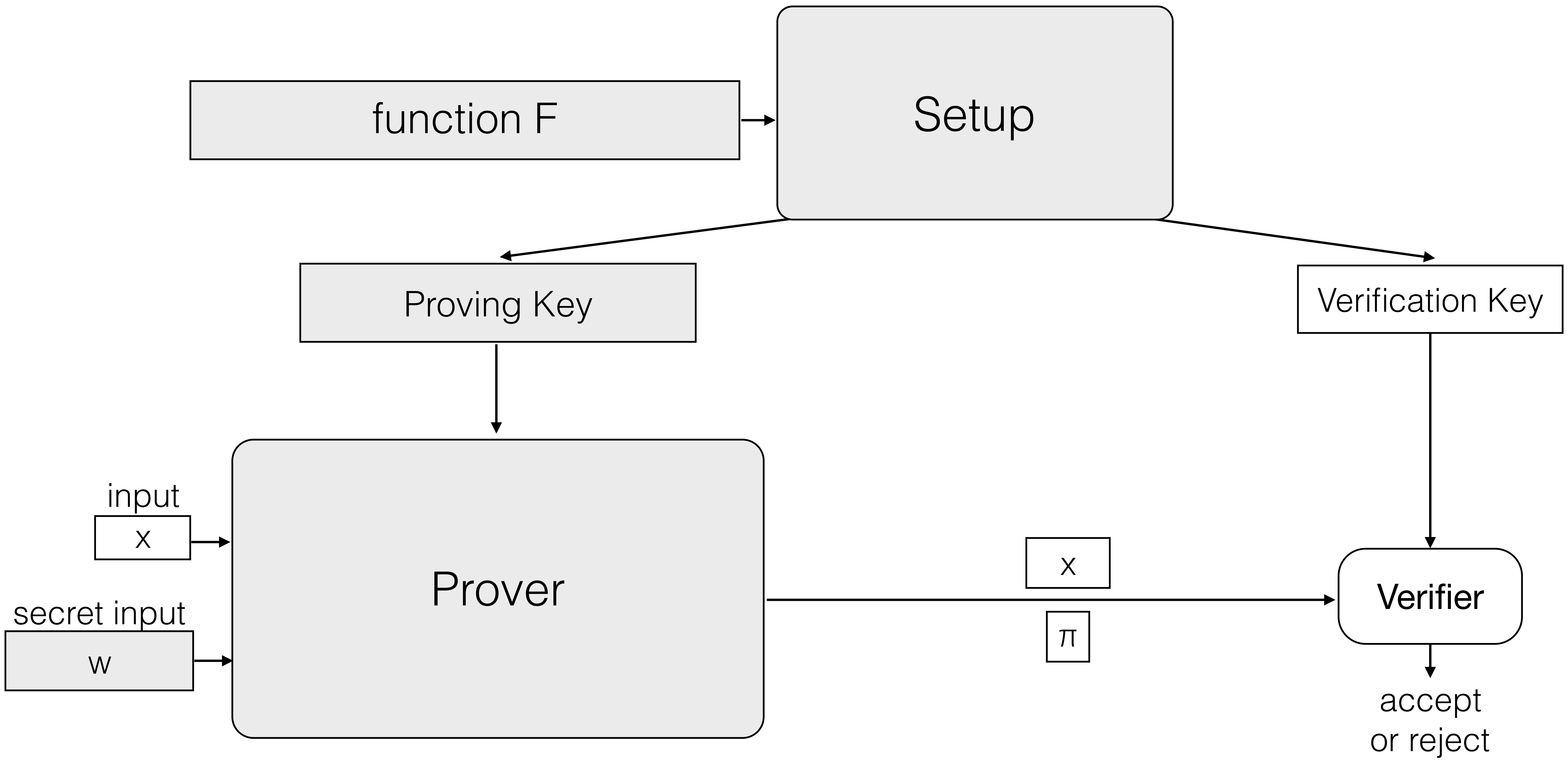
Our Approach

MONOLITHIC
zkSNARK
[Groth16]

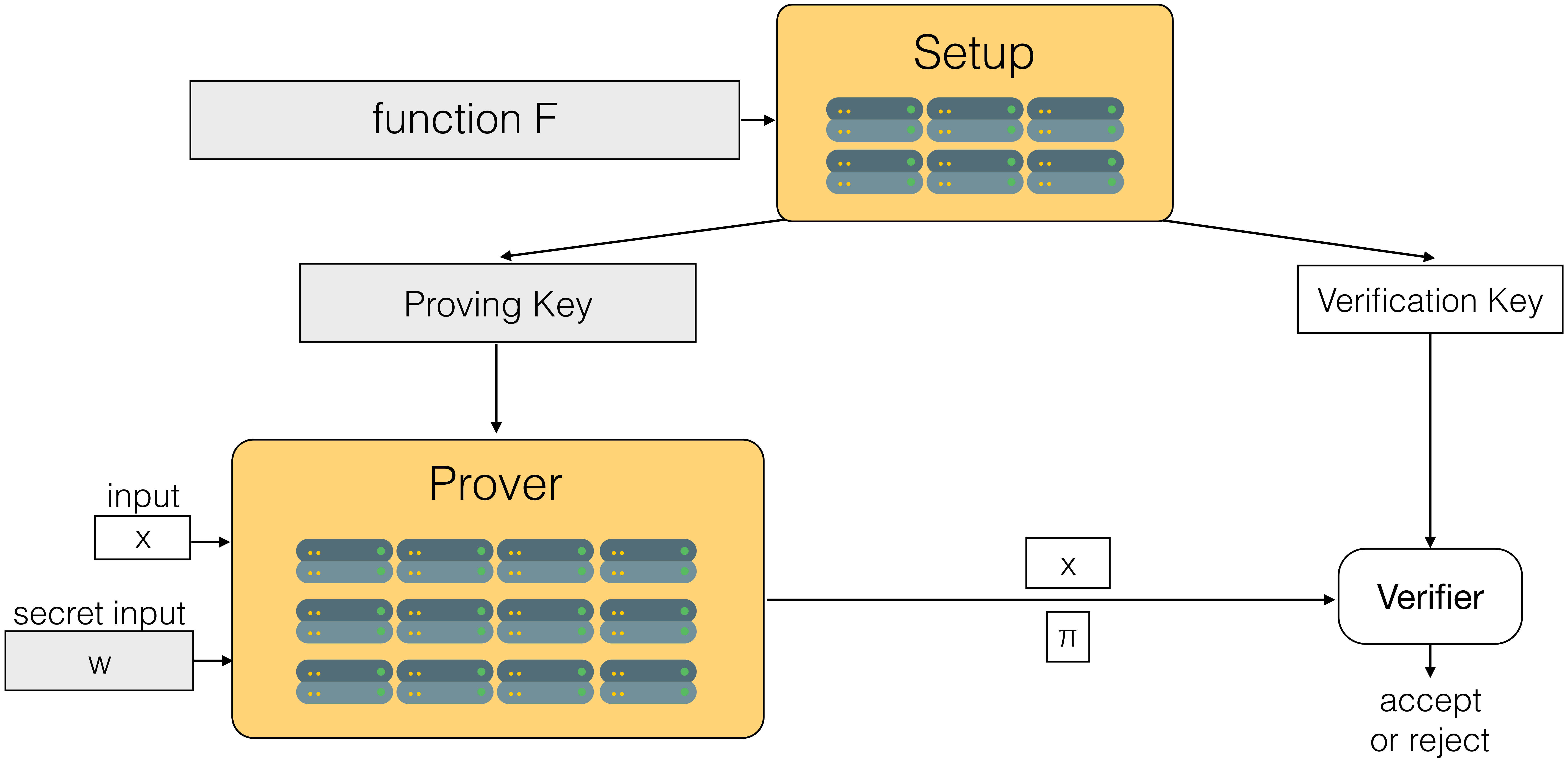
Our Approach



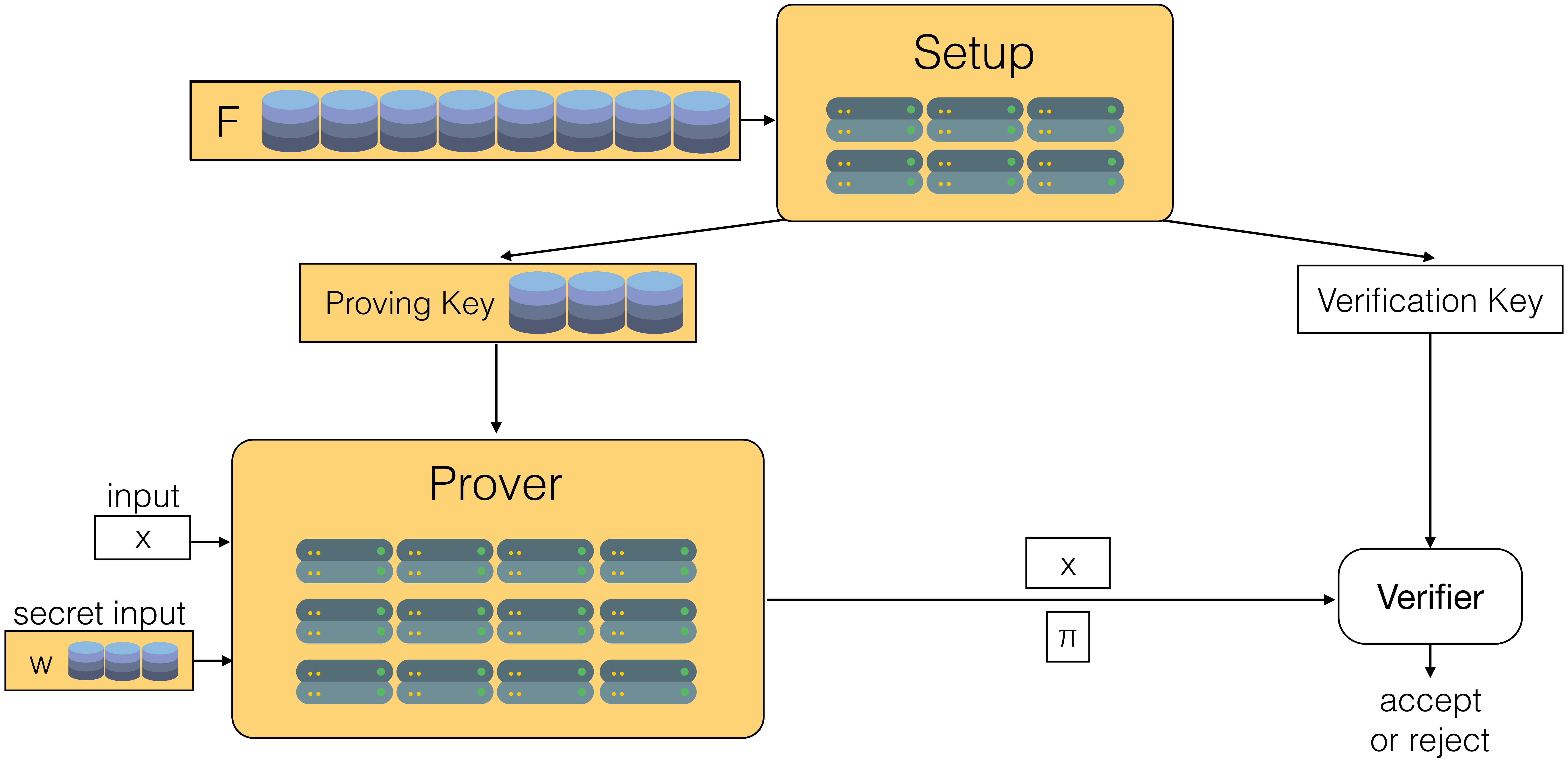
Challenges


function F
Setup
Proving Key
Verification Key
input
x
secret input
w
Prover
x
π
Verifier
accept
or reject

Challenges

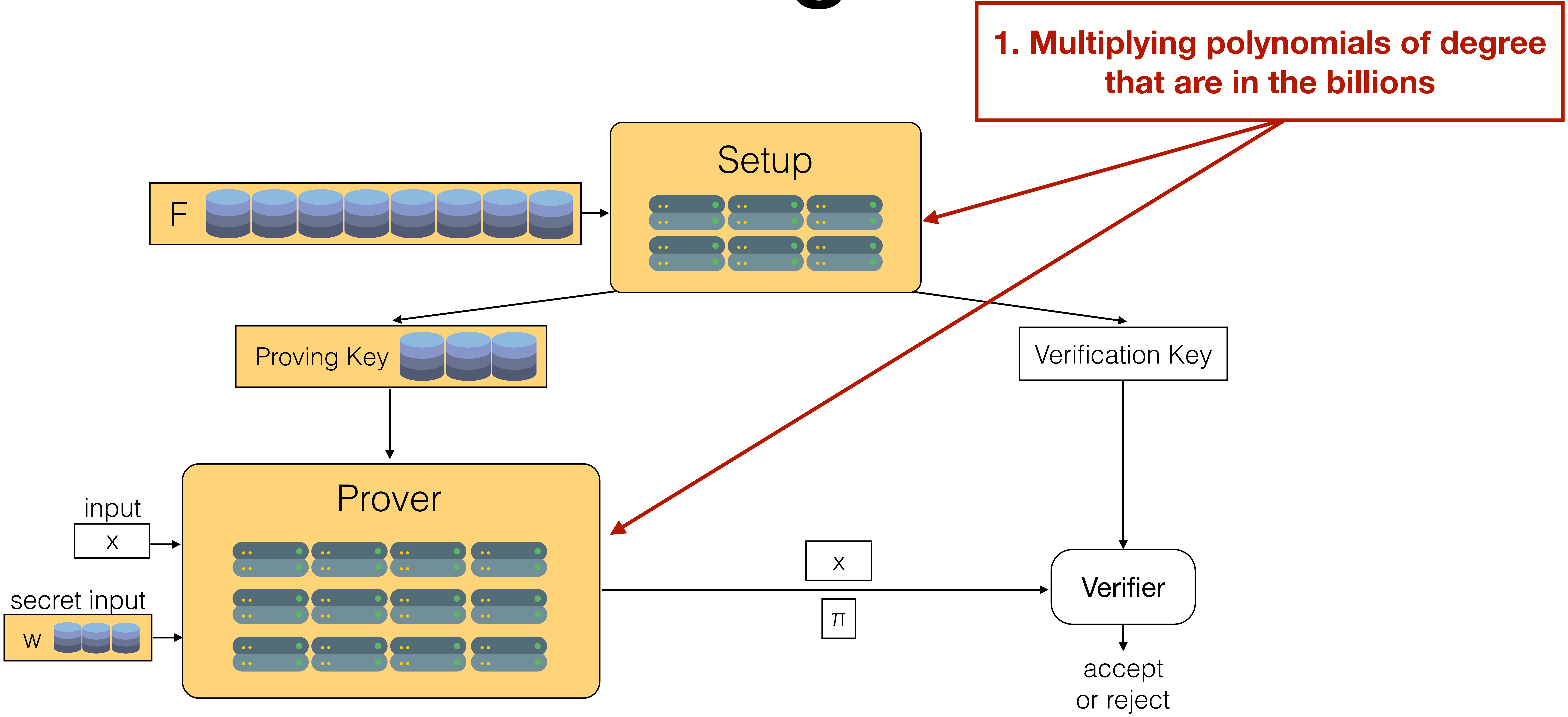

Setup
function F
Proving Key
Verification Key
input
x
Prover
secret input
w
x
π
Verifier
accept
or reject

Challenges



Setup
F
Proving Key
Verification Key
Prover
input
x
secret input
w
x
π
Verifier
accept
or reject

Challenges

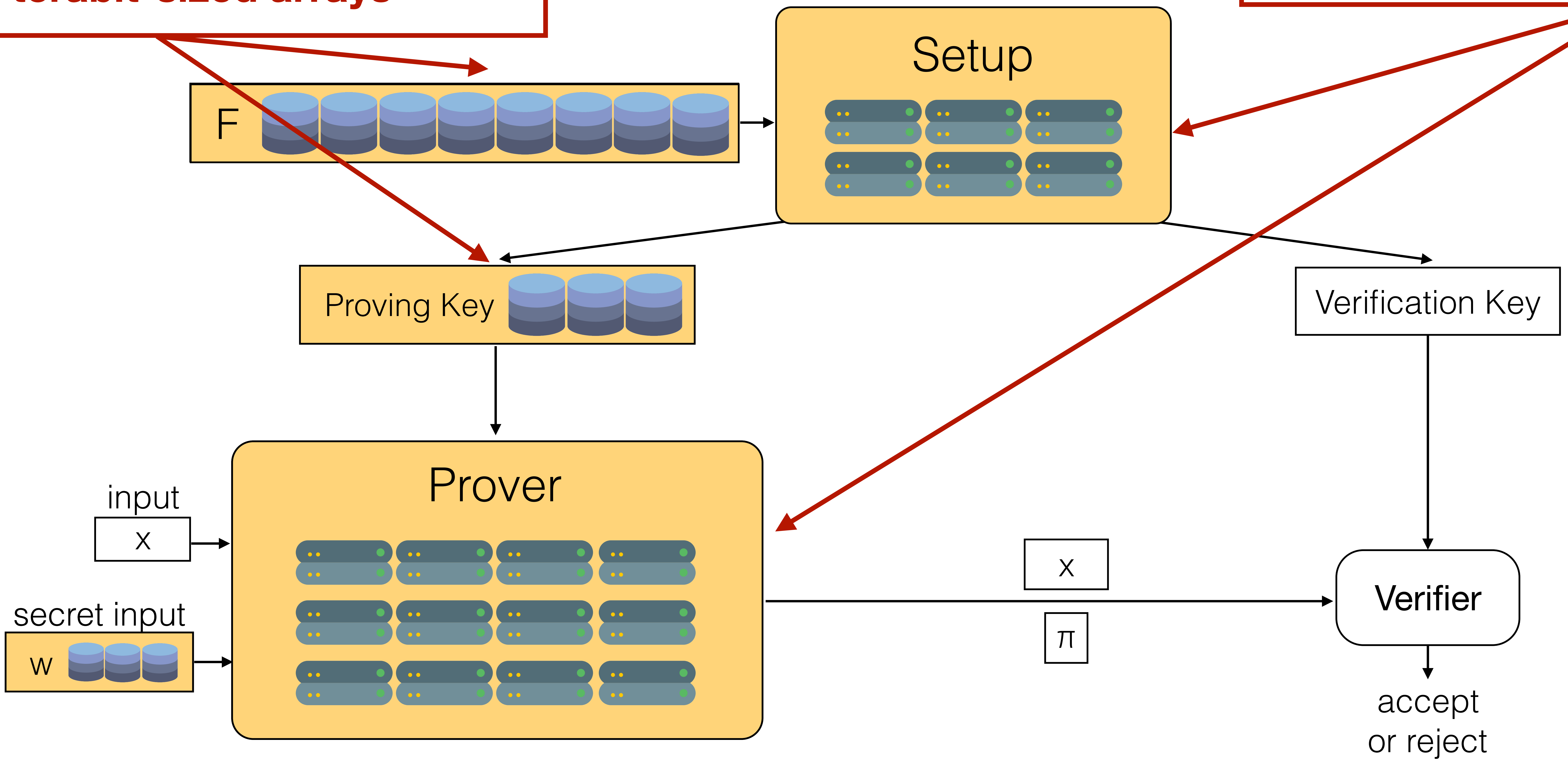


1. Multiplying polynomials of degree that are in the billions
Setup
F
Proving Key
Verification Key
Prover
input
x
secret input
w
x
π
Verifier
accept
or reject

Challenges

2. Representing these polynomials as terabit-sized arrays

1. Multiplying polynomials of degree that are in the billions



Challenges

2. Representing these polynomials as terabit-sized arrays
1. Multiplying polynomials of degree that are in the billions
F
Setup
3. Accessing large pools of shared memory in complex patterns
Proving Key
Verification Key
input
x
Prover
secret input
w
x
π
Verifier
accept or reject

Challenges

2. Representing these polynomials as terabit-sized arrays
1. Multiplying polynomials of degree that are in the billions
F
Setup
3. Accessing large pools of shared memory in complex patterns
Proving Key
Verification Key
4. Synchronizing shared state that incurs significant network delays
input
x
Prover
secret input
w
x
π
Verifier
accept or reject

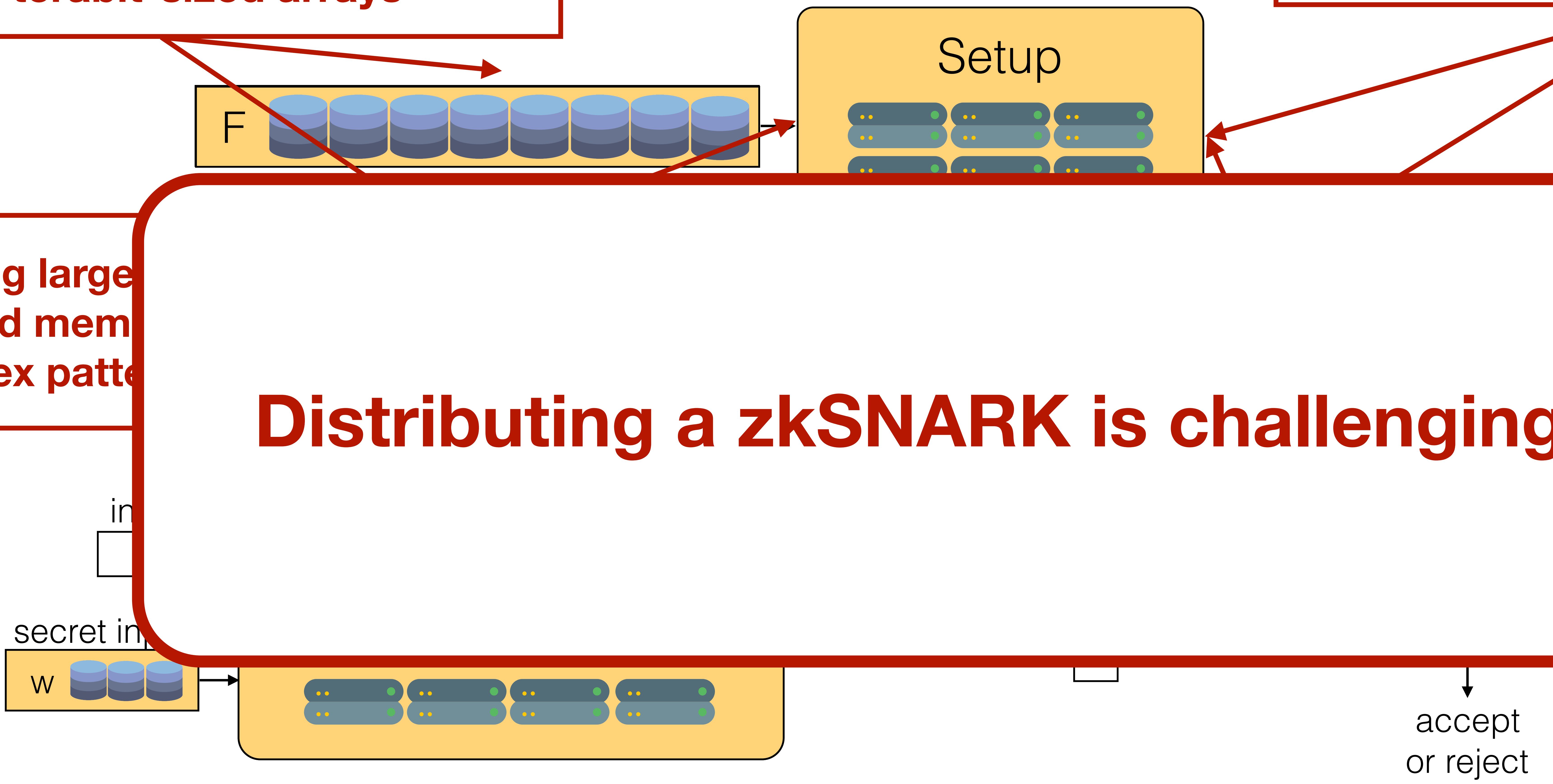
Challenges

2. Representing these polynomials as terabit-sized arrays

1. Multiplying polynomials of degree that are in the billions

3. Accessing large of shared mem in complex patte

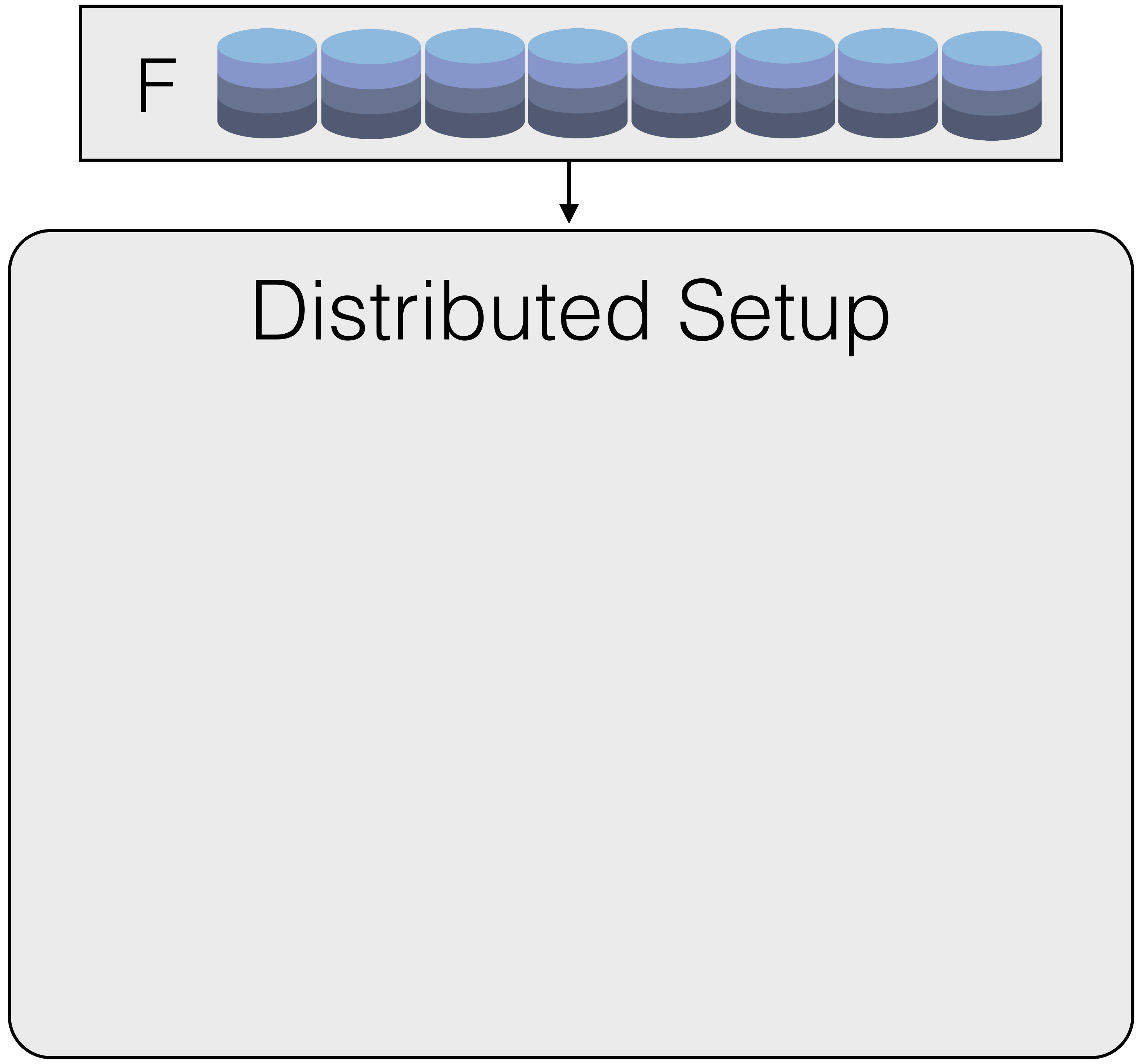
shared state that nt network delays



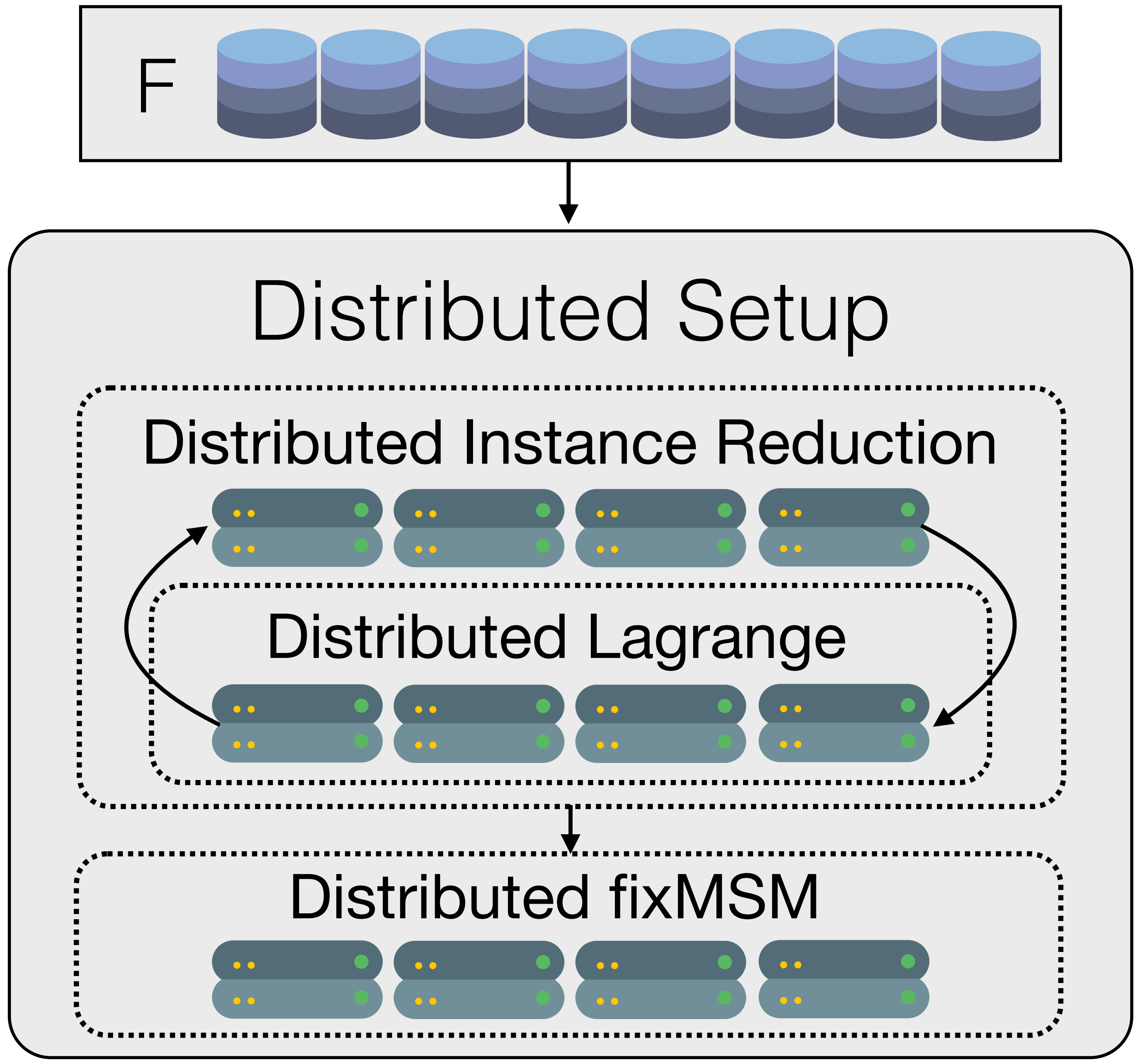
Distributing a zkSNARK is challenging

DIZK Architecture

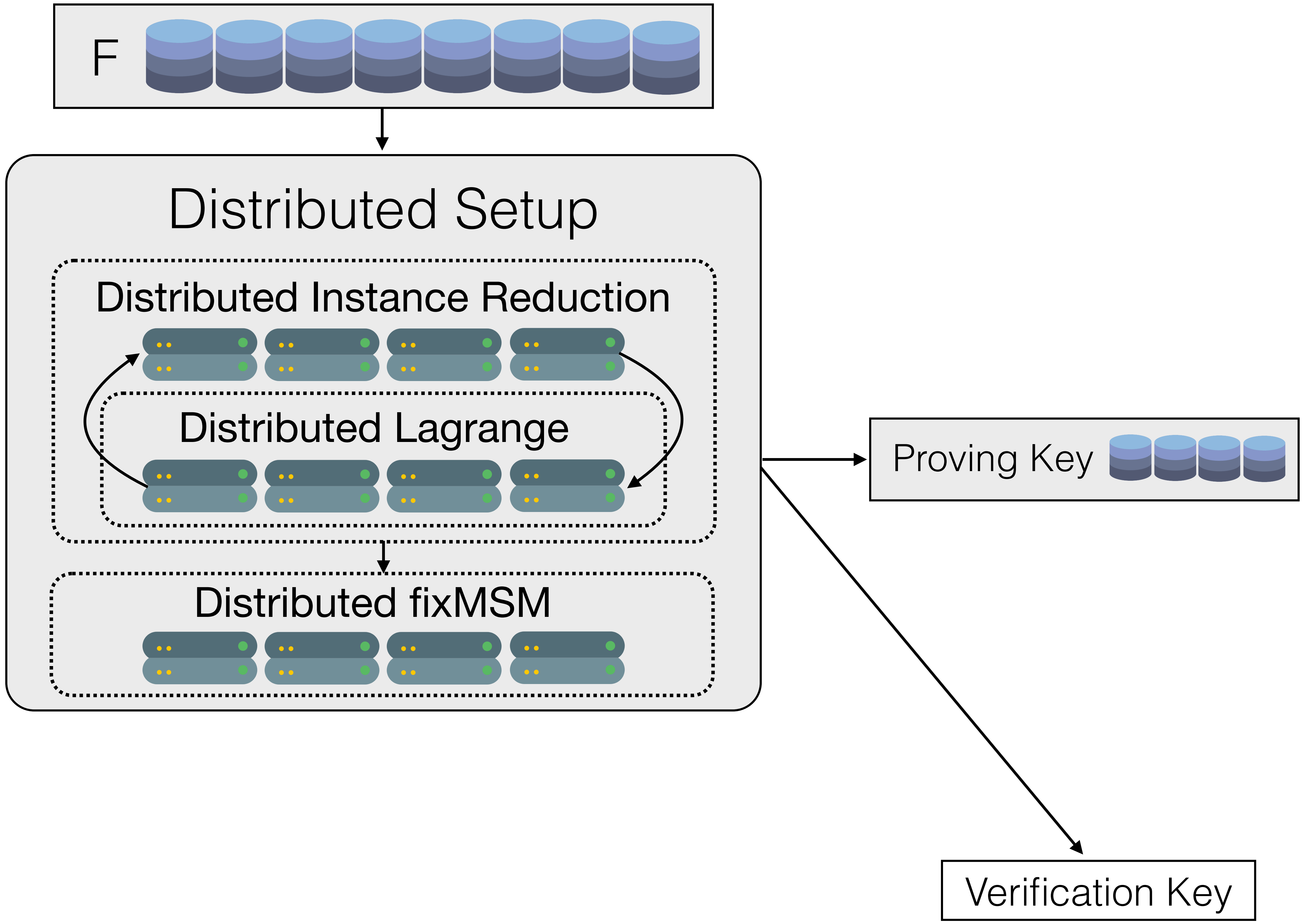
DIZK Architecture



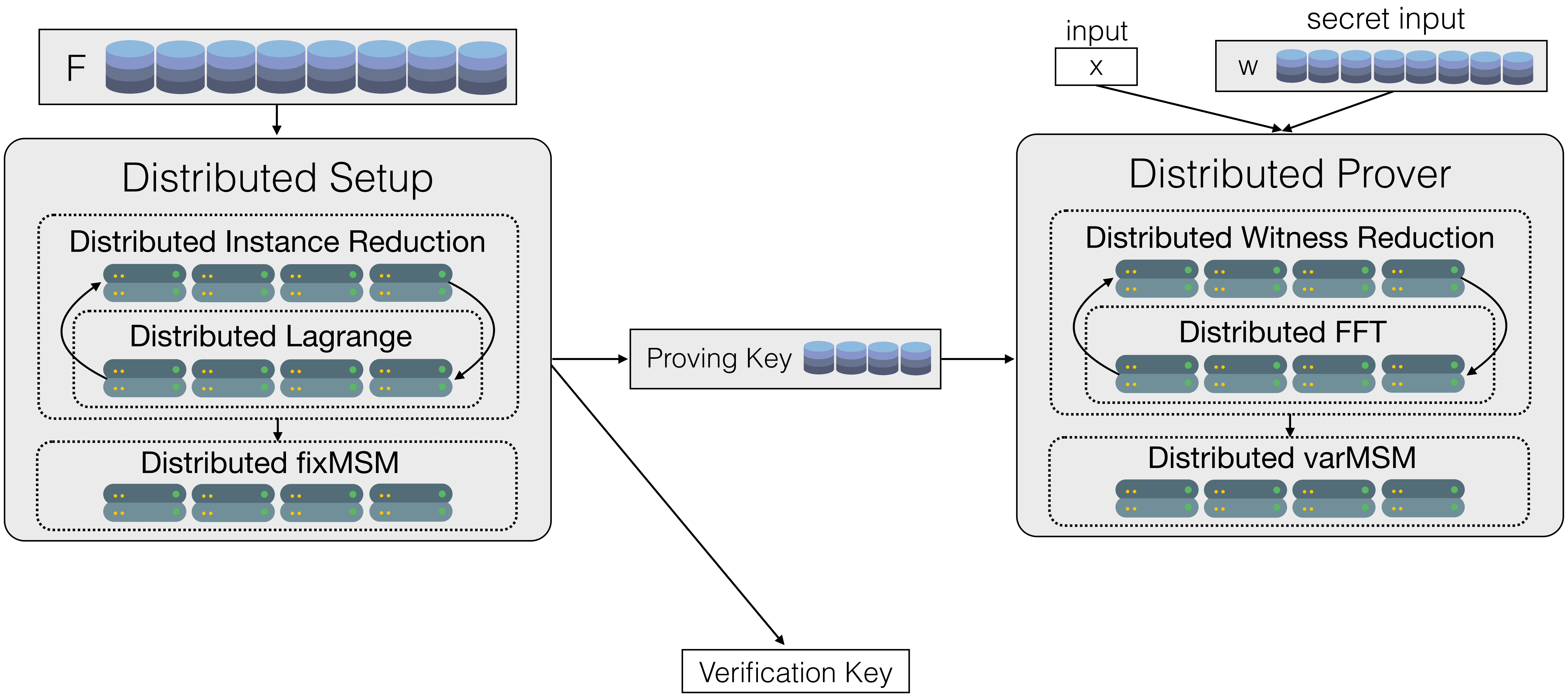
DIZK Architecture



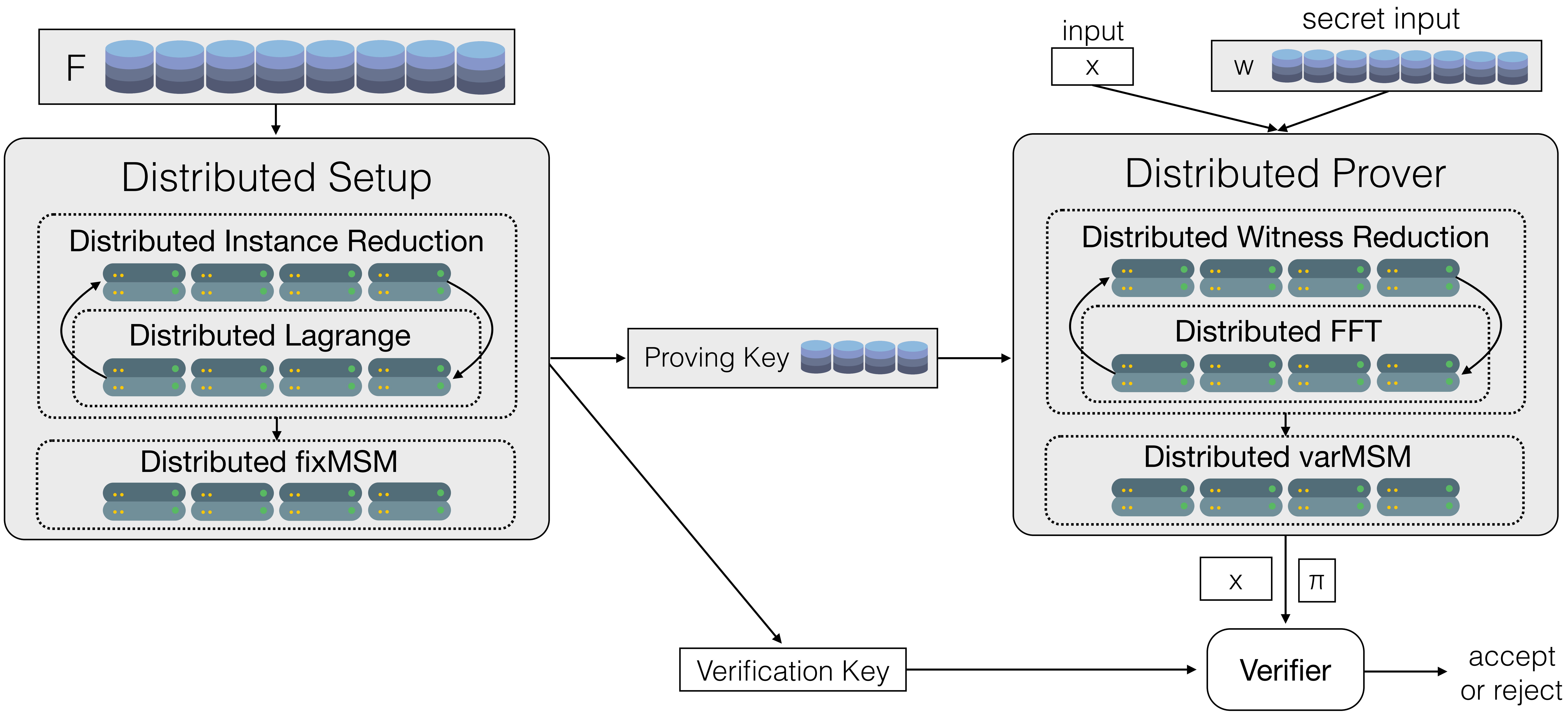
DIZK Architecture



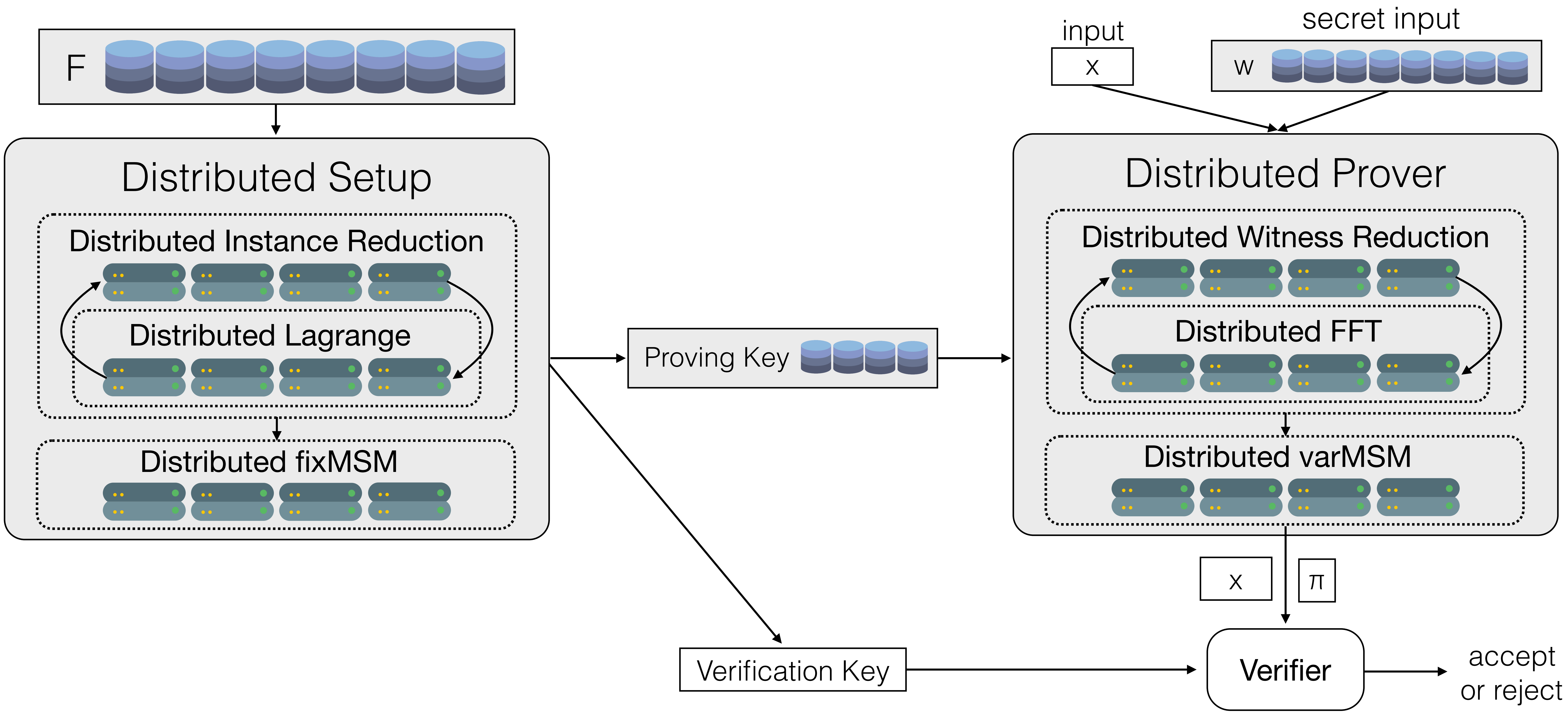
DIZK Architecture



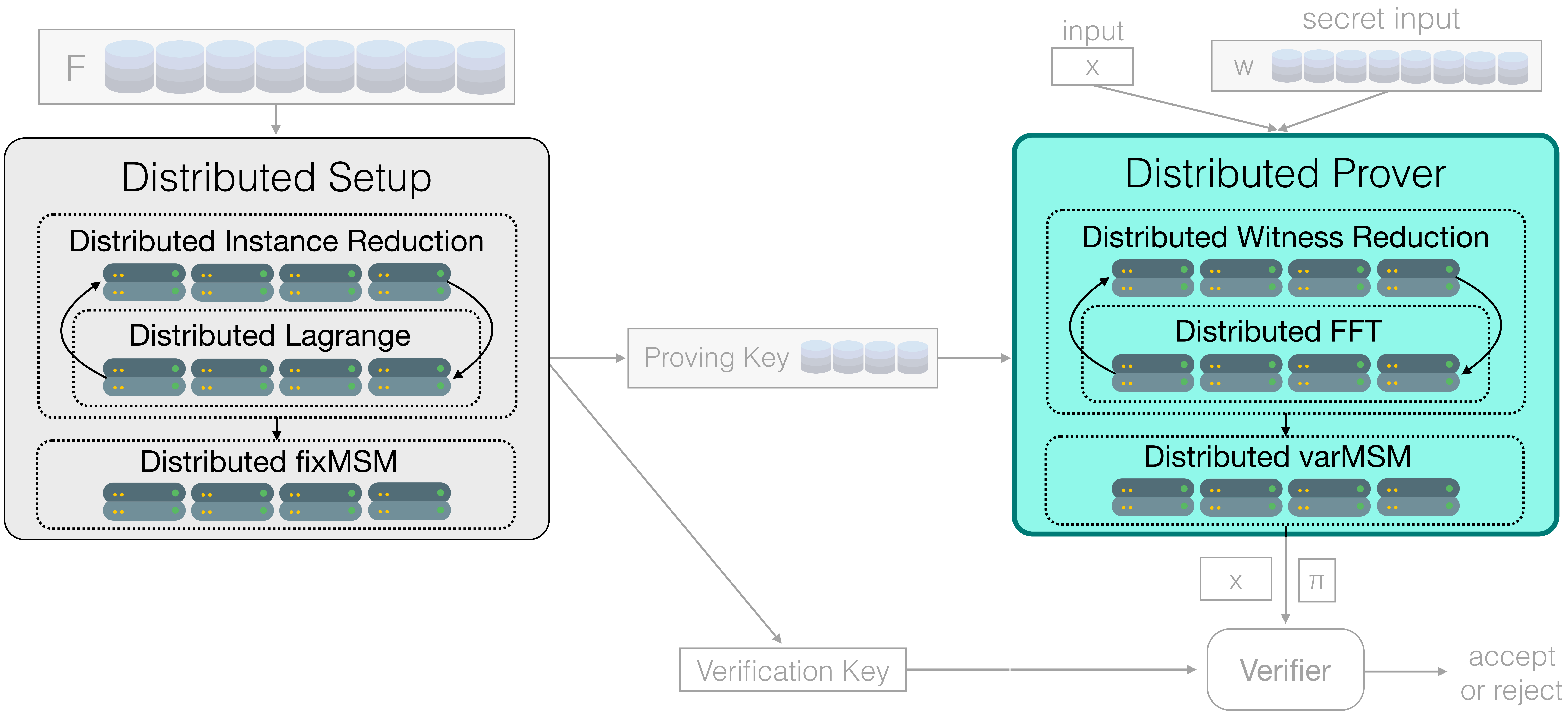
DIZK Architecture



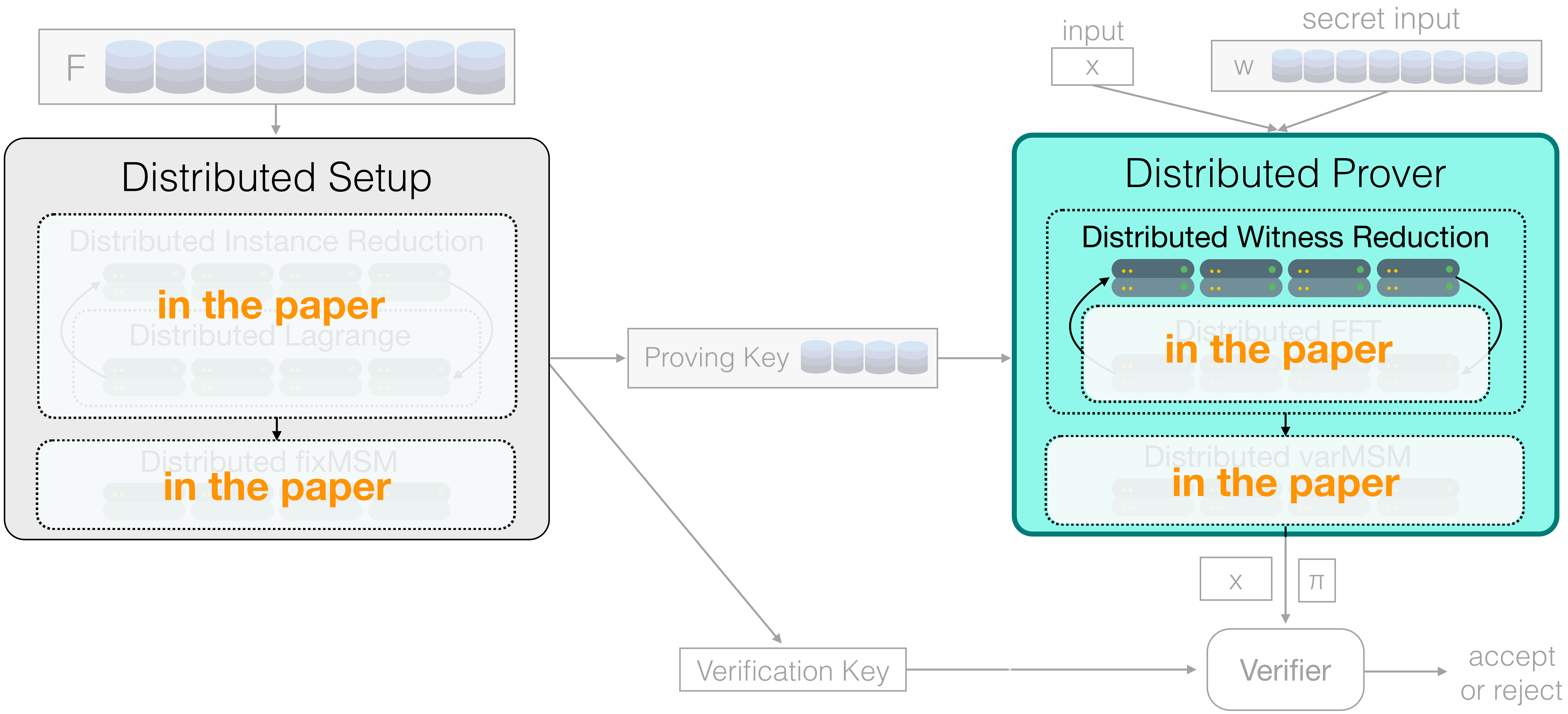
DIZK Architecture



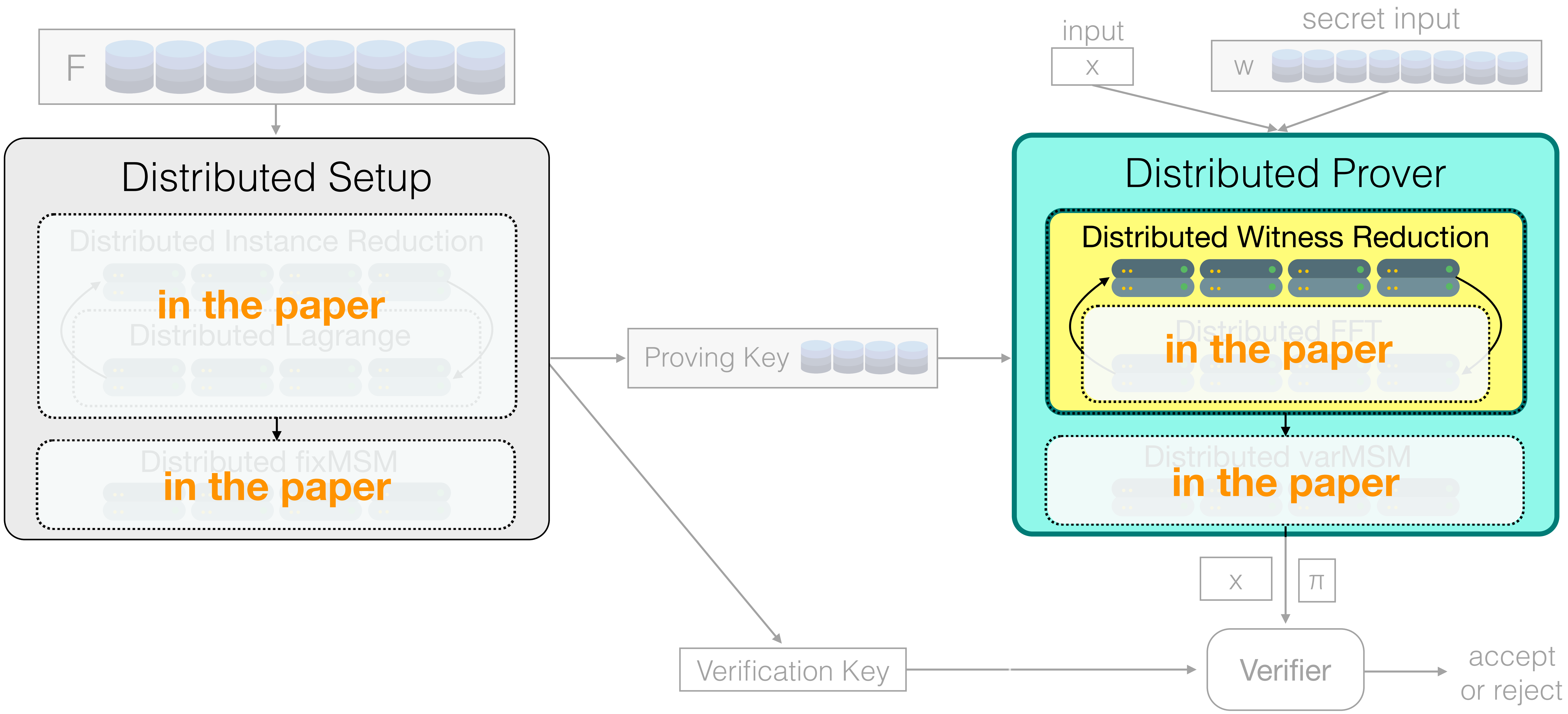
DIZK Architecture



DIZK Architecture



DIZK Architecture



Witness Reduction

[GGPR 13]

Witness Reduction

[GGPR 13]

Billion gate **circuit** → Billion degree **polynomial**

Witness Reduction

[GGPR 13]

Billion gate **circuit** → Billion degree **polynomial**

$$\boxed{H(X) = \frac{\left(\sum_{i=0}^{N} A_i(X)\, z_i\right) \cdot \left(\sum_{i=0}^{N} B_i(X)\, z_i\right) - \left(\sum_{i=0}^{N} C_i(X)\, z_i\right)}{Z_D(X)}}$$

Witness Reduction

[GGPR 13]

Billion gate **circuit** → Billion degree **polynomial**

$$H(X) = \frac{\boxed{\left(\sum_{i=0}^{N} A_i(X)\, z_i\right)} \cdot \boxed{\left(\sum_{i=0}^{N} B_i(X)\, z_i\right)} - \boxed{\left(\sum_{i=0}^{N} C_i(X)\, z_i\right)}}{Z_D(X)}$$

Witness Reduction

[GGPR 13]

Billion gate **circuit** → Billion degree **polynomial**

N = 10^9

$$H(X) = \frac{\boxed{\left(\sum_{i=0}^{N} A_i(X)\, z_i\right)} \cdot \boxed{\left(\sum_{i=0}^{N} B_i(X)\, z_i\right)} - \boxed{\left(\sum_{i=0}^{N} C_i(X)\, z_i\right)}}{Z_D(X)}$$

Witness Reduction

[GGPR 13]

Billion gate **circuit** → Billion degree **polynomial**

N = 10^9

$$H(X) = \frac{\left(\sum_{i=0}^{N} A_i(X)\, z_i\right) \cdot \left(\sum_{i=0}^{N} B_i(X)\, z_i\right) - \left(\sum_{i=0}^{N} C_i(X)\, z_i\right)}{Z_D(X)}$$

Witness Reduction

[GGPR 13]

Billion gate **circuit** → Billion degree **polynomial**

N = 10^9

matrix $\mathbf{A} = (\mathbf{A}_0, \ldots, \mathbf{A}_N)$

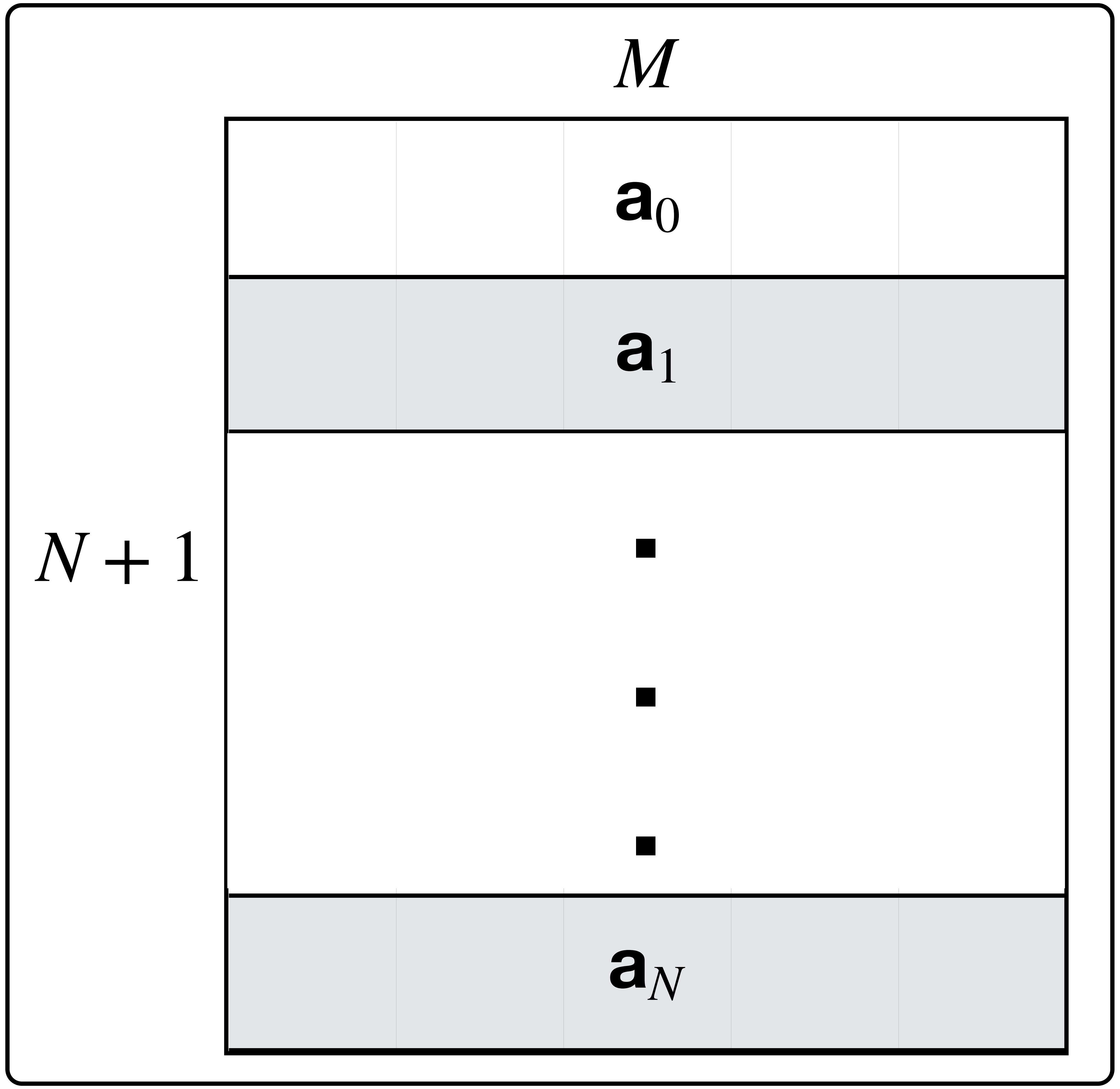
z = vector of N + 1 field elements

$$H(X) = \frac{\boxed{\left(\sum_{i=0}^{N} A_i(X)\, z_i\right)} \cdot \left(\sum_{i=0}^{N} B_i(X)\, z_i\right) - \left(\sum_{i=0}^{N} C_i(X)\, z_i\right)}{Z_D(X)}$$

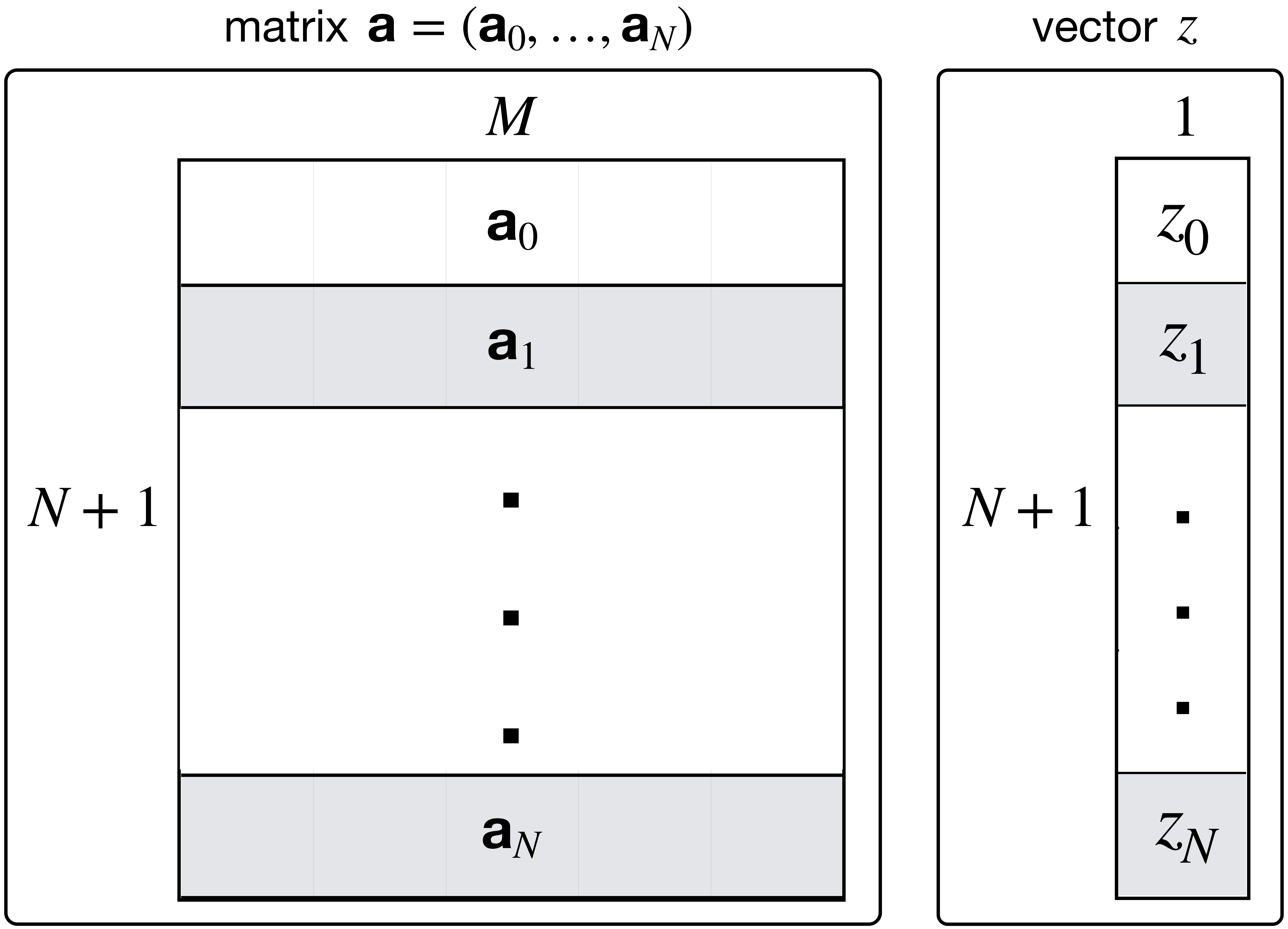
Strawman for $\sum_{i=0}^{N} A_i(X)\ z_i$

Strawman for $\sum_{i=0}^{N} A_i(X)\ z_i$

matrix $\mathbf{a} = (\mathbf{a}_0, \ldots, \mathbf{a}_N)$

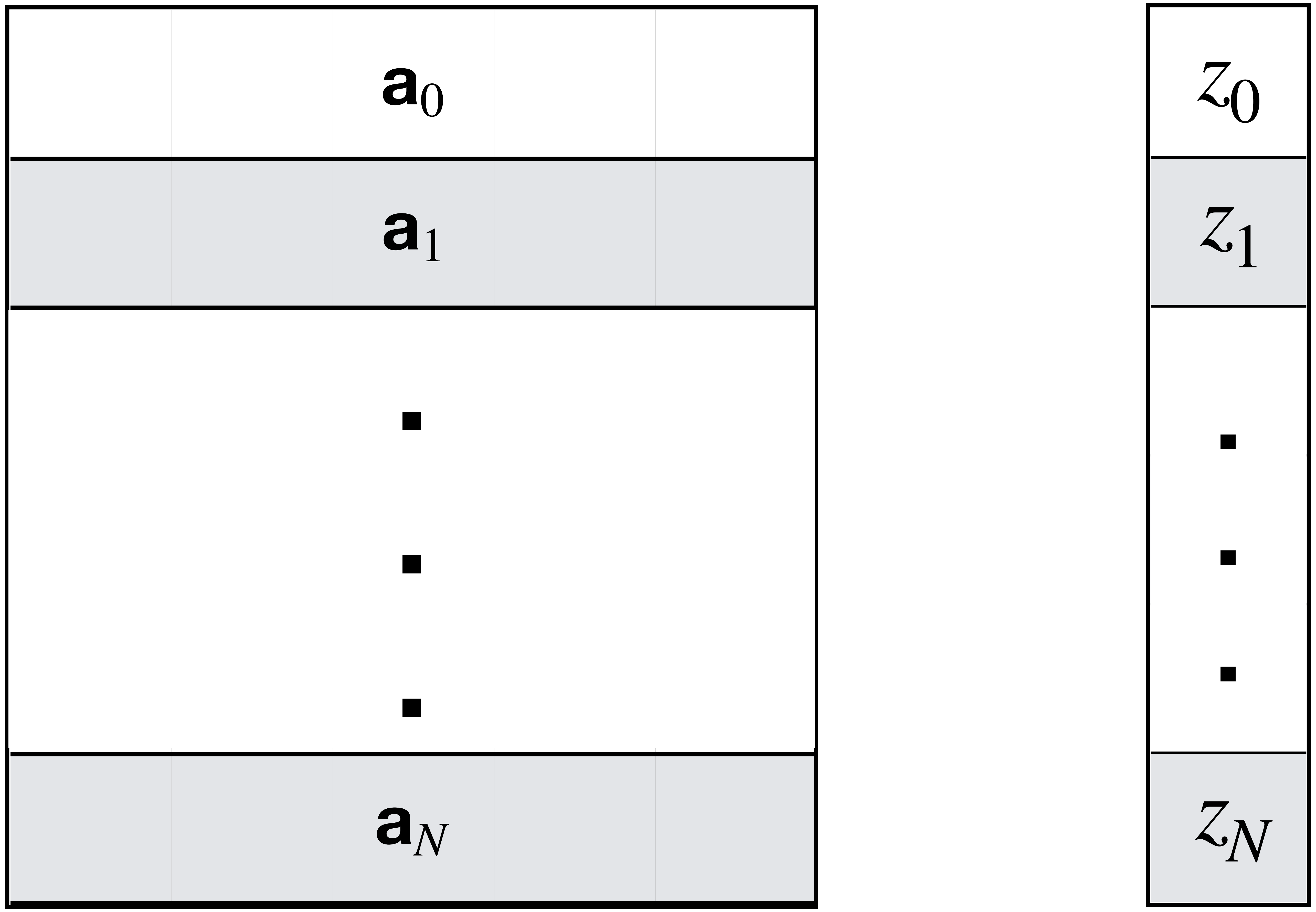


Strawman for $\sum_{i=0}^{N} A_i(X)\, z_i$

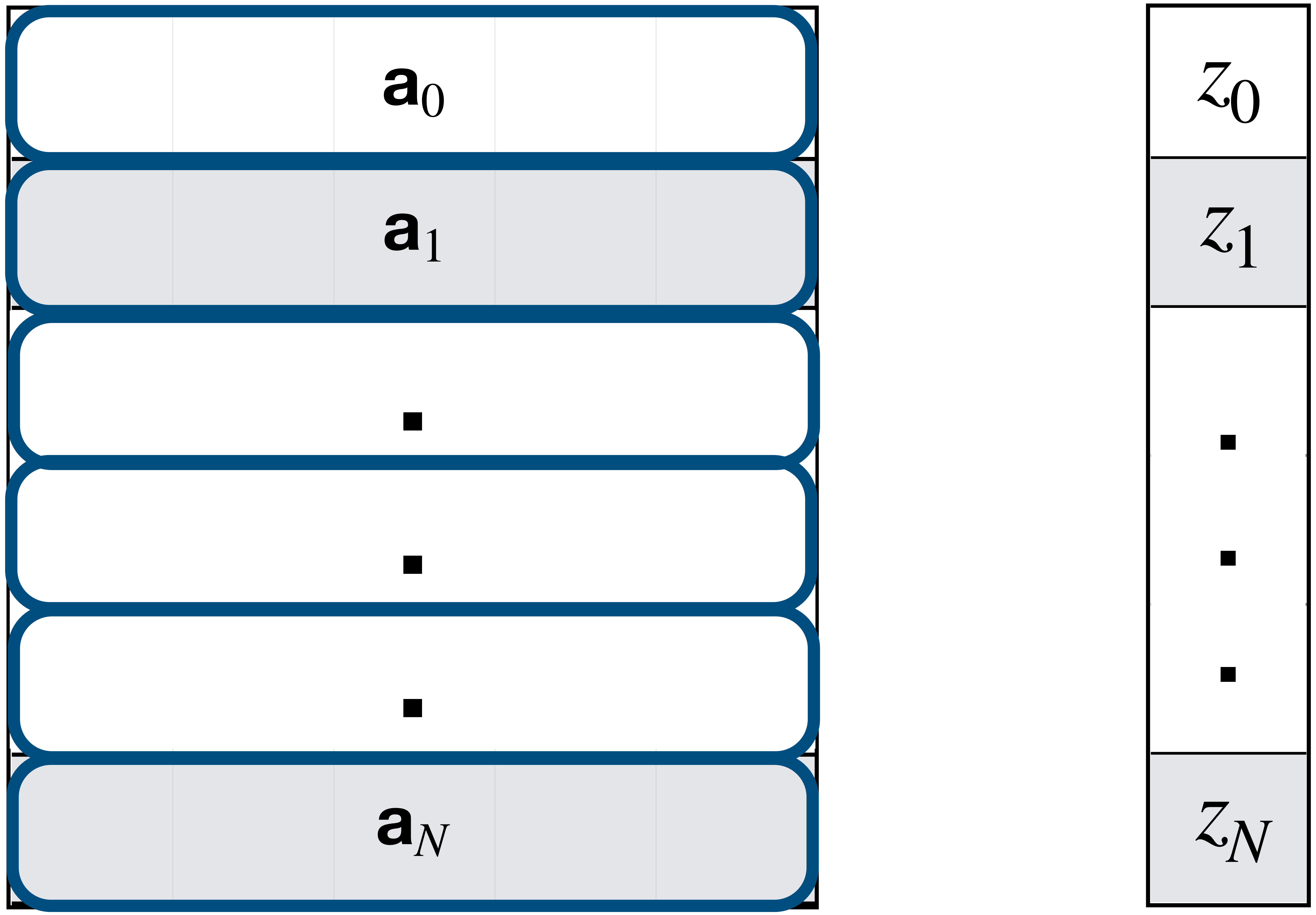


Strawman for $\sum_{i=0}^{N} A_i(X)\ z_i$

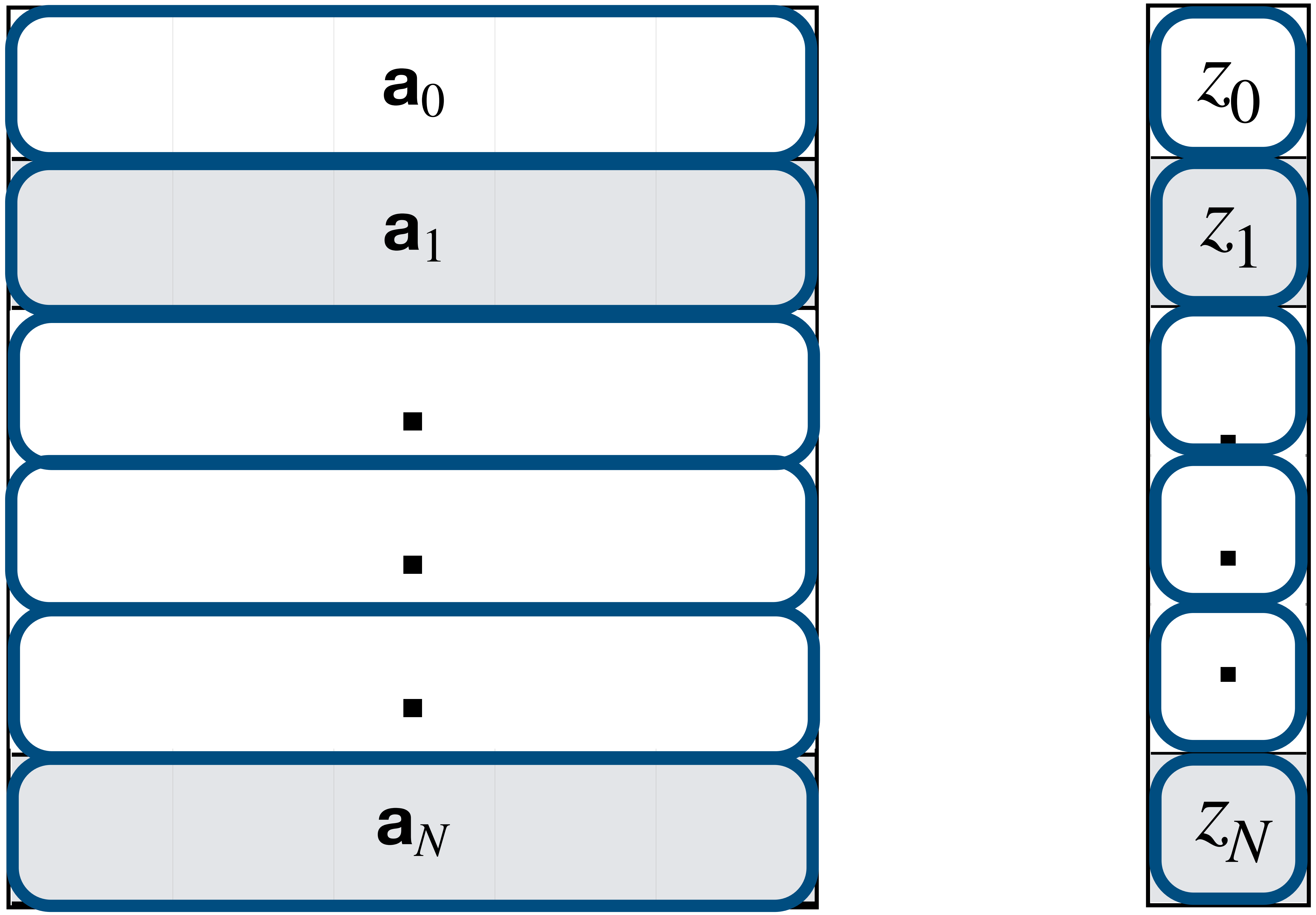
Strawman for $\sum_{i=0}^{N} A_i(X)\ z_i$



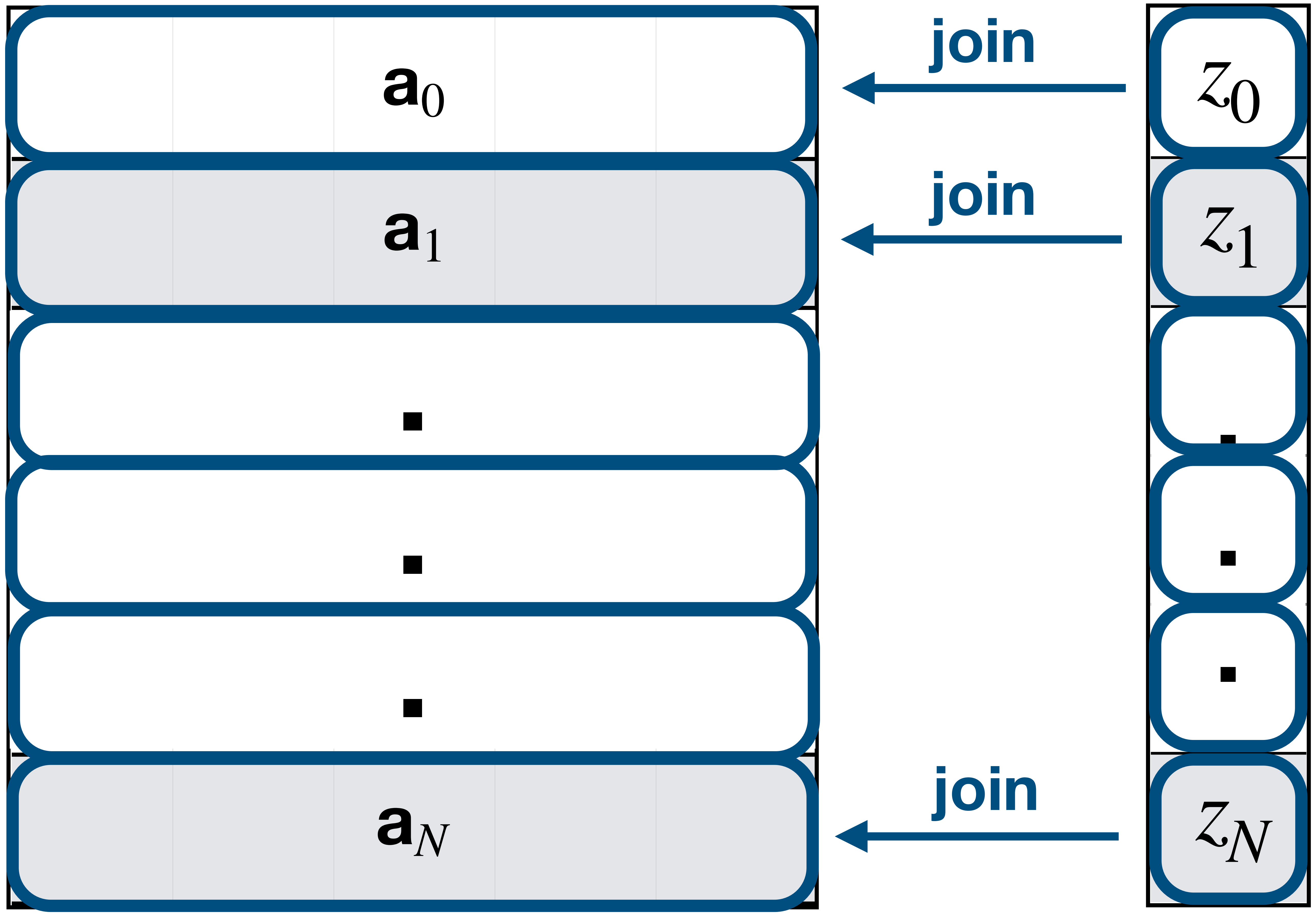
Strawman for $\sum_{i=0}^{N} A_i(X)\ z_i$



Strawman for $\sum_{i=0}^{N} A_i(X)\ z_i$



Strawman for $\sum_{i=0}^{N} A_i(X)\ z_i$



Strawman for $\sum_{i=0}^{N} A_i(X)\ z_i$

Strawman for $\sum_{i=0}^{N} A_i(X)\ z_i$

$(\mathbf{a}_i,\ z_i)$ pairs

$(\mathbf{a}_{0,0}, z_0)$	$(\mathbf{a}_{0,1}, z_0)$	$(\mathbf{a}_{0,2}, z_0)$	$\cdots$	$(\mathbf{a}_{0,M}, z_0)$
$(\mathbf{a}_{1,0}, z_1)$	$(\mathbf{a}_{1,1}, z_1)$	$(\mathbf{a}_{1,2}, z_1)$	$\cdots$	$(\mathbf{a}_{1,M}, z_1)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$(\mathbf{a}_{N,0}, z_N)$	$(\mathbf{a}_{N,1}, z_N)$	$(\mathbf{a}_{N,2}, z_N)$	$\cdots$	$(\mathbf{a}_{N,M}, z_N)$

Strawman for $\sum_{i=0}^{N} A_i(X)\, z_i$

$(991, z_0)$	$(681, z_0)$	$(1978, z_0)$	$\cdots$	$(517, z_0)$
$(0, z_1)$	$(2476, z_1)$	$(0, z_1)$	$\cdots$	$(0, z_1)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$(0, z_N)$	$(8629, z_N)$	$(0, z_N)$	$\cdots$	$(0, z_N)$

Strawman for $\sum_{i=0}^{N} A_i(X)\ z_i$

Almost sparse

$(991, z_0)$	$(681, z_0)$	$(1978, z_0)$	$\cdots$	$(517, z_0)$
$(0, z_1)$	$(2476, z_1)$	$(0, z_1)$	$\cdots$	$(0, z_1)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$(0, z_N)$	$(8629, z_N)$	$(0, z_N)$	$\cdots$	$(0, z_N)$

Strawman for $\sum_{i=0}^{N} A_i(X)\, z_i$

Almost sparse

$(991, z_0)$	$(681, z_0)$	$(1978, z_0)$	$\cdots$	$(517, z_0)$
$(0, z_1)$	$(2476, z_1)$	$(0, z_1)$	$\cdots$	$(0, z_1)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$(0, z_N)$	$(8629, z_N)$	$(0, z_N)$	$\cdots$	$(0, z_N)$

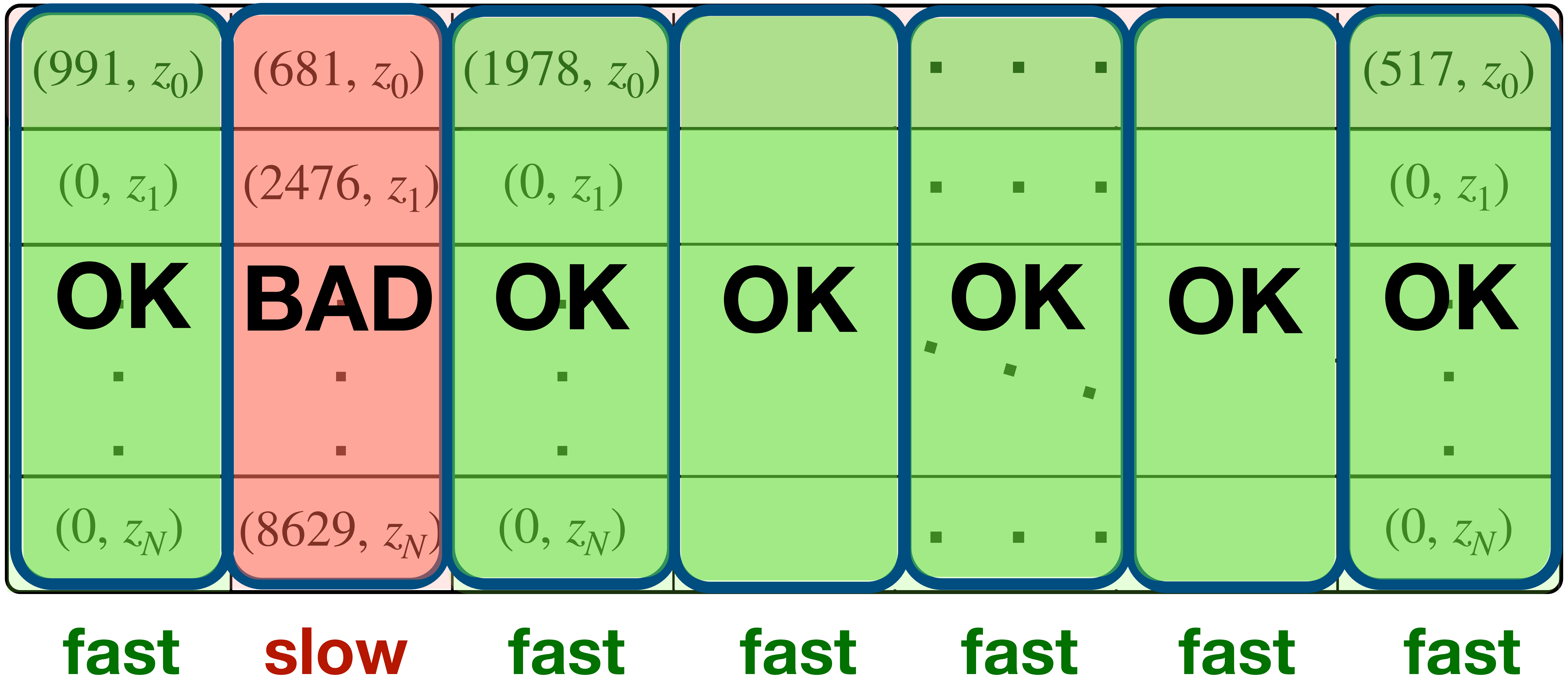
Strawman for $\sum_{i=0}^{N} A_i(X)\, z_i$

Almost sparse

$(991, z_0)$	$(681, z_0)$	$(1978, z_0)$		$\cdots$		$(517, z_0)$
$(0, z_1)$	$(2476, z_1)$	$(0, z_1)$		$\cdots$		$(0, z_1)$
$\vdots$	$\vdots$	$\vdots$		$\ddots$		$\vdots$
$(0, z_N)$	$(8629, z_N)$	$(0, z_N)$		$\cdots$		$(0, z_N)$

Strawman for $\sum_{i=0}^{N} A_i(X)\ z_i$

Almost sparse



Strawman for $\sum_{i=0}^{N} A_i(X)\, z_i$

Almost sparse

$(991, z_0)$	$(681, z_0)$	$(1978, z_0)$	$\cdots$	$(517, z_0)$
$(0, z_1)$	$(2476, z_1)$	$(0, z_1)$	$\cdots$	$(0, z_1)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$(0, z_N)$	$(8629, z_N)$	$(0, z_N)$	$\cdots$	$(0, z_N)$

Strawman for $\sum_{i=0}^{N} A_i(X)\ z_i$

Almost sparse

$(991, z_0)$	$(681, z_0)$	$(1978, z_0)$	$\cdots$	$(517, z_0)$
$(0, z_1)$	$(2476, z_1)$	$(0, z_1)$	$\cdots$	$(0, z_1)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$(0, z_N)$	$(8629, z_N)$	$(0, z_N)$	$\cdots$	$(0, z_N)$

Strawman for $\sum_{i=0}^{N} A_i(X)\, z_i$

Almost sparse

$(991, z_0)$	$(681, z_0)$	$(1978, z_0)$	**BAD** $\cdots$	$(517, z_0)$	**slow**
$(0, z_1)$	$(2476, z_1)$	$(0, z_1)$	**OK** $\cdots$	$(0, z_1)$	**fast**
$\vdots$	$\vdots$	$\vdots$	**OK** $\ddots$	$\vdots$	**fast**
			OK		**fast**
$(0, z_N)$	$(8629, z_N)$	$(0, z_N)$	**OK** $\cdots$	$(0, z_N)$	**fast**

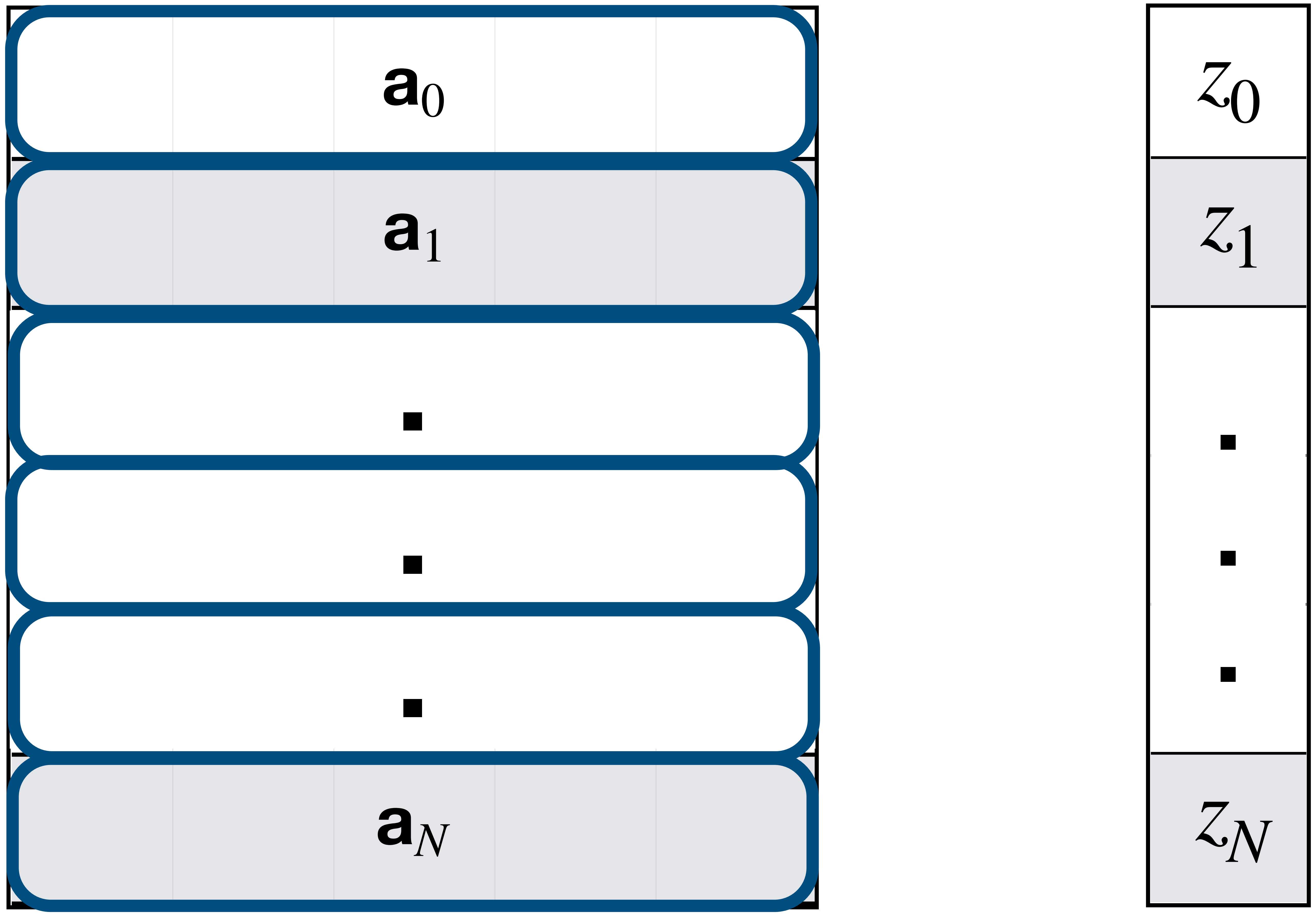
Off-the-shelf Approaches

Off-the-shelf Approaches

Replicate and ***partition*** the data
so that the computation is **distributed evenly**.

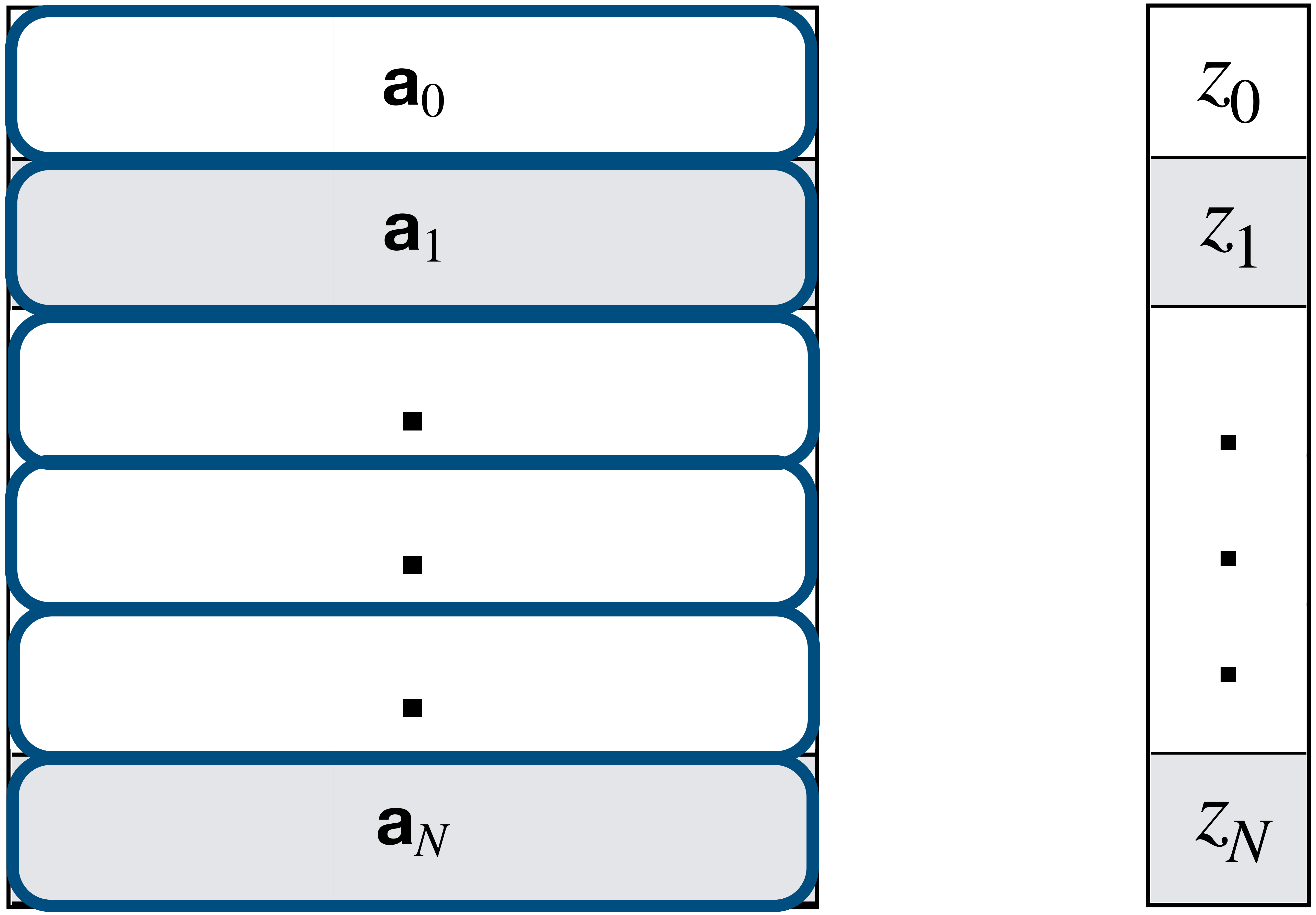
blockjoin

(Common technique to address data skew)



blockjoin

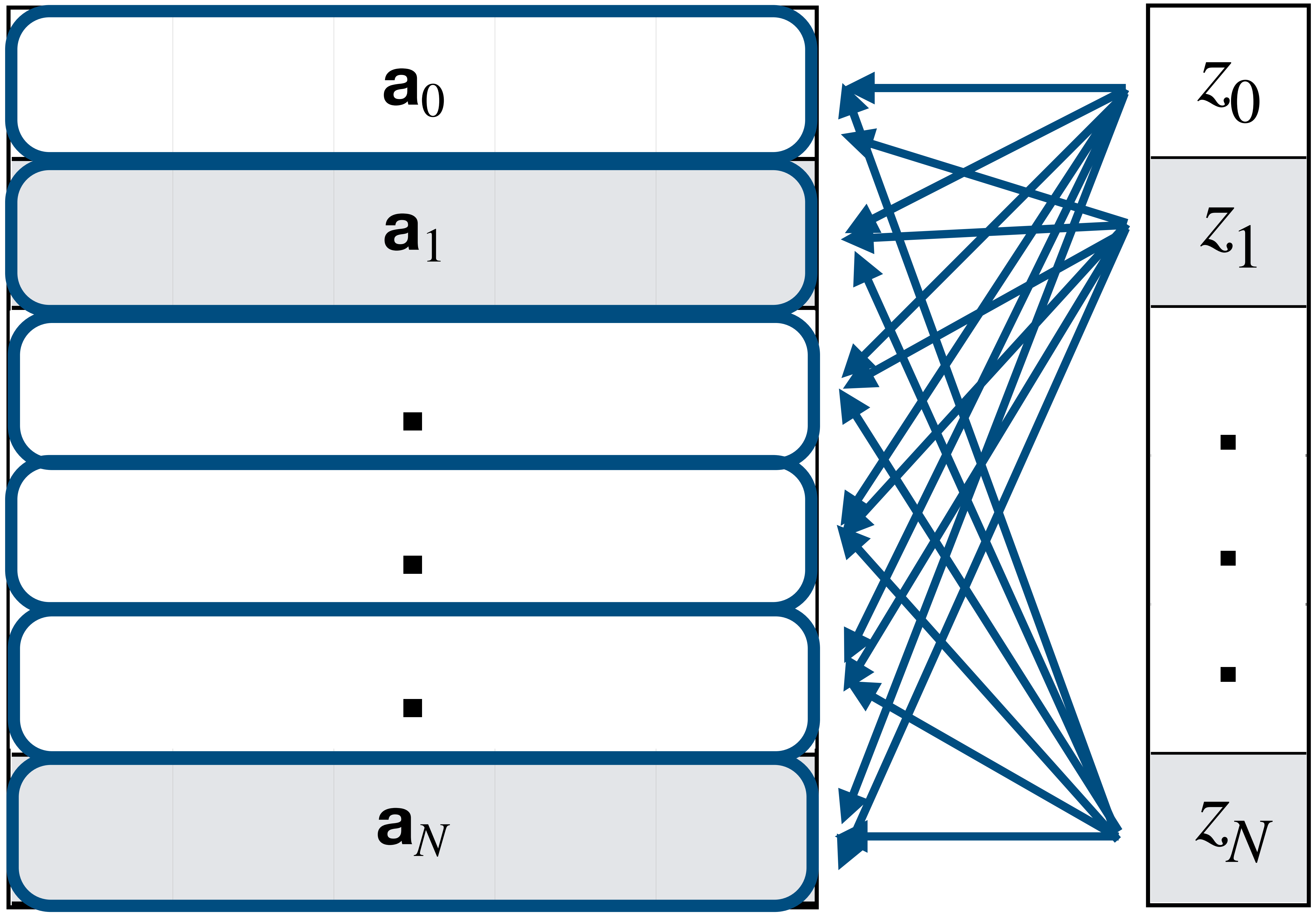
(Common technique to address data skew)



Replicated each entry for *every* machine

blockjoin

(Common technique to address data skew)



Replicated each entry for *every* machine

`blockjoin`

(Common technique to address data skew)

$(\mathbf{a}_{0,0}, z_0)$	$(\mathbf{a}_{0,1}, z_0)$	$(\mathbf{a}_{0,2}, z_0)$	$(\mathbf{a}_{0,3}, z_0)$	$\cdots$	$(\mathbf{a}_{0,M-1}, z_0)$	$(\mathbf{a}_{0,M}, z_0)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$
$(\mathbf{a}_{0,0}, z_N)$	$(\mathbf{a}_{0,1}, z_N)$	$(\mathbf{a}_{0,2}, z_N)$	$(\mathbf{a}_{0,3}, z_N)$	$\cdots$	$(\mathbf{a}_{0,M-1}, z_N)$	$(\mathbf{a}_{0,M}, z_N)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$(\mathbf{a}_{N,0}, z_0)$	$(\mathbf{a}_{N,1}, z_0)$	$(\mathbf{a}_{N,2}, z_0)$	$(\mathbf{a}_{N,3}, z_0)$	$\cdots$	$(\mathbf{a}_{N,M-1}, z_0)$	$(\mathbf{a}_{N,M}, z_0)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$		
$(\mathbf{a}_{N,0}, z_N)$	$(\mathbf{a}_{N,1}, z_N)$	$(\mathbf{a}_{N,2}, z_N)$	$(\mathbf{a}_{N,3}, z_N)$	$\cdots$	$(\mathbf{a}_{N,M-1}, z_0)$	$(\mathbf{a}_{N,M}, z_N)$

`blockjoin` (N + 1) * (# partitions) replications

(Common technique to address data skew)

$(\mathbf{a}_{0,0}, z_0)$	$(\mathbf{a}_{0,1}, z_0)$	$(\mathbf{a}_{0,2}, z_0)$	$(\mathbf{a}_{0,3}, z_0)$	$\cdots$	$(\mathbf{a}_{0,M-1}, z_0)$	$(\mathbf{a}_{0,M}, z_0)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$
$(\mathbf{a}_{0,0}, z_N)$	$(\mathbf{a}_{0,1}, z_N)$	$(\mathbf{a}_{0,2}, z_N)$	$(\mathbf{a}_{0,3}, z_N)$	$\cdots$	$(\mathbf{a}_{0,M-1}, z_N)$	$(\mathbf{a}_{0,M}, z_N)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$(\mathbf{a}_{N,0}, z_0)$	$(\mathbf{a}_{N,1}, z_0)$	$(\mathbf{a}_{N,2}, z_0)$	$(\mathbf{a}_{N,3}, z_0)$	$\cdots$	$(\mathbf{a}_{N,M-1}, z_0)$	$(\mathbf{a}_{N,M}, z_0)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$		
$(\mathbf{a}_{N,0}, z_N)$	$(\mathbf{a}_{N,1}, z_N)$	$(\mathbf{a}_{N,2}, z_N)$	$(\mathbf{a}_{N,3}, z_N)$	$\cdots$	$(\mathbf{a}_{N,M-1}, z_0)$	$(\mathbf{a}_{N,M}, z_N)$

blockjoin

(Common technique to address data skew)

Billions

(N + 1) * (# partitions) replications

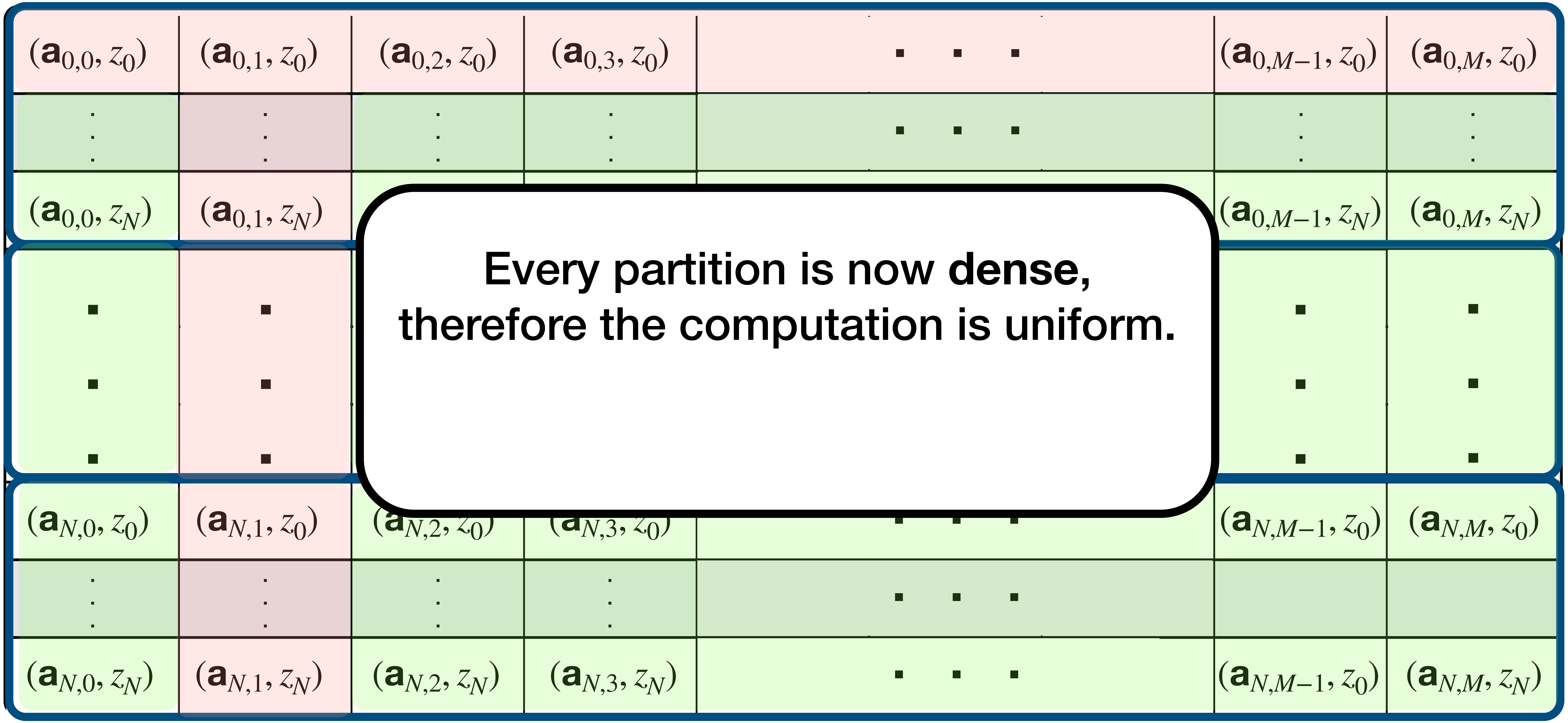
$(\mathbf{a}_{0,0}, z_0)$	$(\mathbf{a}_{0,1}, z_0)$	$(\mathbf{a}_{0,2}, z_0)$	$(\mathbf{a}_{0,3}, z_0)$	$\cdots$	$(\mathbf{a}_{0,M-1}, z_0)$	$(\mathbf{a}_{0,M}, z_0)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$
$(\mathbf{a}_{0,0}, z_N)$	$(\mathbf{a}_{0,1}, z_N)$	$(\mathbf{a}_{0,2}, z_N)$	$(\mathbf{a}_{0,3}, z_N)$	$\cdots$	$(\mathbf{a}_{0,M-1}, z_N)$	$(\mathbf{a}_{0,M}, z_N)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$(\mathbf{a}_{N,0}, z_0)$	$(\mathbf{a}_{N,1}, z_0)$	$(\mathbf{a}_{N,2}, z_0)$	$(\mathbf{a}_{N,3}, z_0)$	$\cdots$	$(\mathbf{a}_{N,M-1}, z_0)$	$(\mathbf{a}_{N,M}, z_0)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$		
$(\mathbf{a}_{N,0}, z_N)$	$(\mathbf{a}_{N,1}, z_N)$	$(\mathbf{a}_{N,2}, z_N)$	$(\mathbf{a}_{N,3}, z_N)$	$\cdots$	$(\mathbf{a}_{N,M-1}, z_0)$	$(\mathbf{a}_{N,M}, z_N)$

blockjoin

(Common technique to address data skew)

Billions

(N + 1) * (# partitions) replications



blockjoin

(Common technique to address data skew)

(N + 1) * (# partitions) replications

Billions

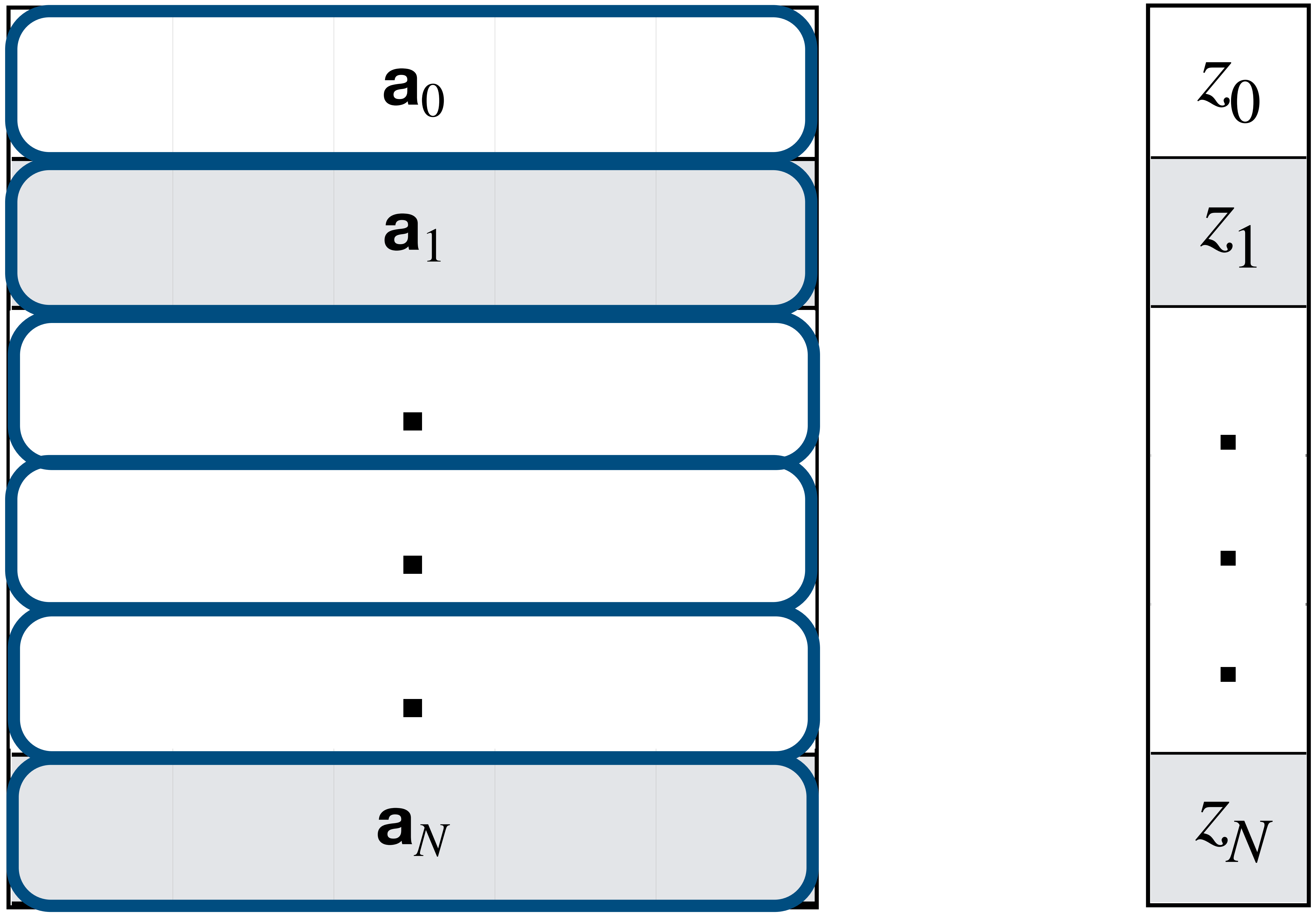
$(\mathbf{a}_{0,0}, z_0)$	$(\mathbf{a}_{0,1}, z_0)$	$(\mathbf{a}_{0,2}, z_0)$	$(\mathbf{a}_{0,3}, z_0)$	$\cdots$	$(\mathbf{a}_{0,M-1}, z_0)$	$(\mathbf{a}_{0,M}, z_0)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$
$(\mathbf{a}_{0,0}, z_N)$	$(\mathbf{a}_{0,1}, z_N)$				$(\mathbf{a}_{0,M-1}, z_N)$	$(\mathbf{a}_{0,M}, z_N)$
$\vdots$	$\vdots$				$\vdots$	$\vdots$
$(\mathbf{a}_{N,0}, z_0)$	$(\mathbf{a}_{N,1}, z_0)$	$(\mathbf{a}_{N,2}, z_0)$	$(\mathbf{a}_{N,3}, z_0)$	$\cdots$	$(\mathbf{a}_{N,M-1}, z_0)$	$(\mathbf{a}_{N,M}, z_0)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$		
$(\mathbf{a}_{N,0}, z_N)$	$(\mathbf{a}_{N,1}, z_N)$	$(\mathbf{a}_{N,2}, z_N)$	$(\mathbf{a}_{N,3}, z_N)$	$\cdots$	$(\mathbf{a}_{N,M-1}, z_0)$	$(\mathbf{a}_{N,M}, z_N)$

Every partition is now **dense**, therefore the computation is uniform.

(However, the table is huge and impractical to compute)

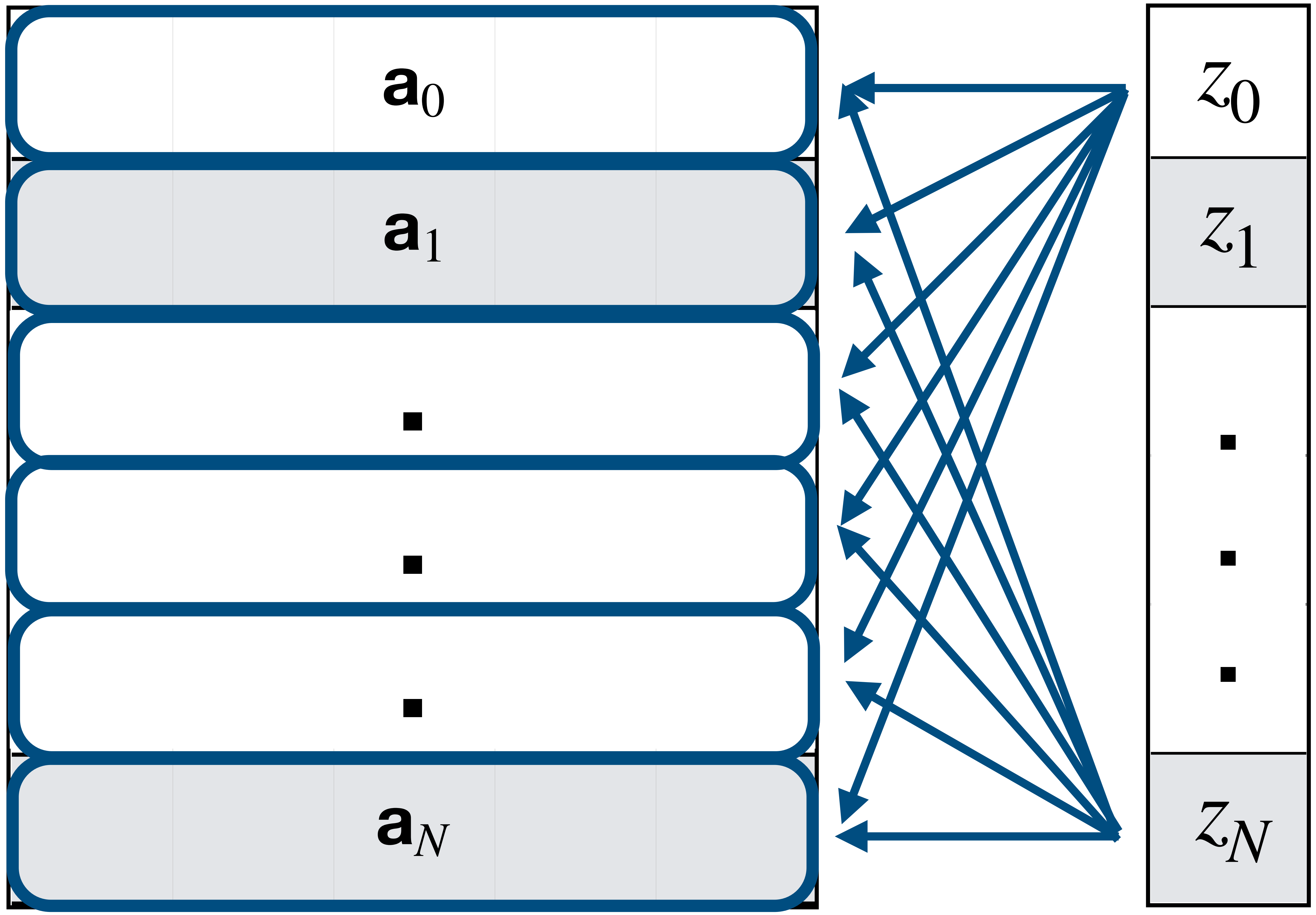
skewjoin

(Another common technique to address data skew. Source: *Tresata*)



skewjoin

(Another common technique to address data skew. Source: *Tresata*)



Compute usage *statistics* and replicate *frequently-used* entries for every machine

skewjoin

(Another common technique to address data skew. Source: *Tresata*)

$(\mathbf{a}_{0,0}, z_0)$	$(\mathbf{a}_{0,1}, z_0)$	$(\mathbf{a}_{0,2}, z_0)$	$\cdots$	$(\mathbf{a}_{0,M}, z_0)$
$(\mathbf{a}_{1,0}, z_1)$	$(\mathbf{a}_{1,1}, z_1)$	$(\mathbf{a}_{1,2}, z_1)$	$\cdots$	$(\mathbf{a}_{1,M}, z_1)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$(\mathbf{a}_{N,0}, z_N)$	$(\mathbf{a}_{N,1}, z_N)$	$(\mathbf{a}_{N,2}, z_N)$	$\cdots$	$(\mathbf{a}_{N,M}, z_N)$

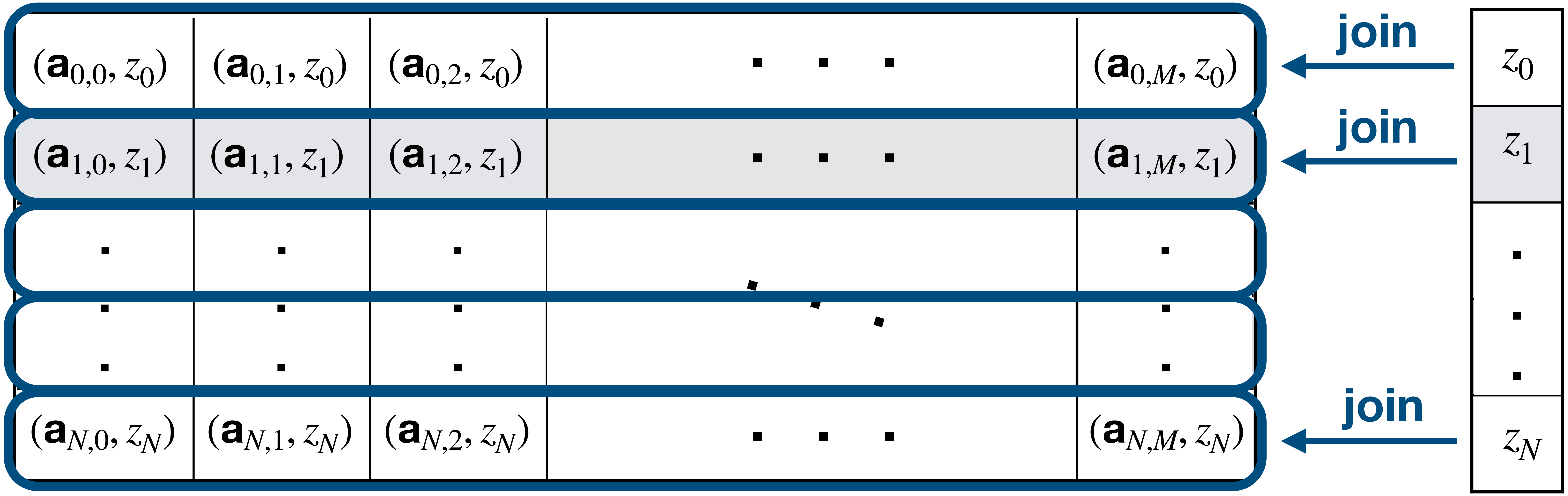
skewjoin

(Another common technique to address data skew. Source: *Tresata*)

$(\mathbf{a}_{0,0}, z_0)$	$(\mathbf{a}_{0,1}, z_0)$	$(\mathbf{a}_{0,2}, z_0)$	$\cdots$	$(\mathbf{a}_{0,M}, z_0)$
$(\mathbf{a}_{1,0}, z_1)$	$(\mathbf{a}_{1,1}, z_1)$	$(\mathbf{a}_{1,2}, z_1)$	$\cdots$	$(\mathbf{a}_{1,M}, z_1)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$(\mathbf{a}_{N,0}, z_N)$	$(\mathbf{a}_{N,1}, z_N)$	$(\mathbf{a}_{N,2}, z_N)$	$\cdots$	$(\mathbf{a}_{N,M}, z_N)$

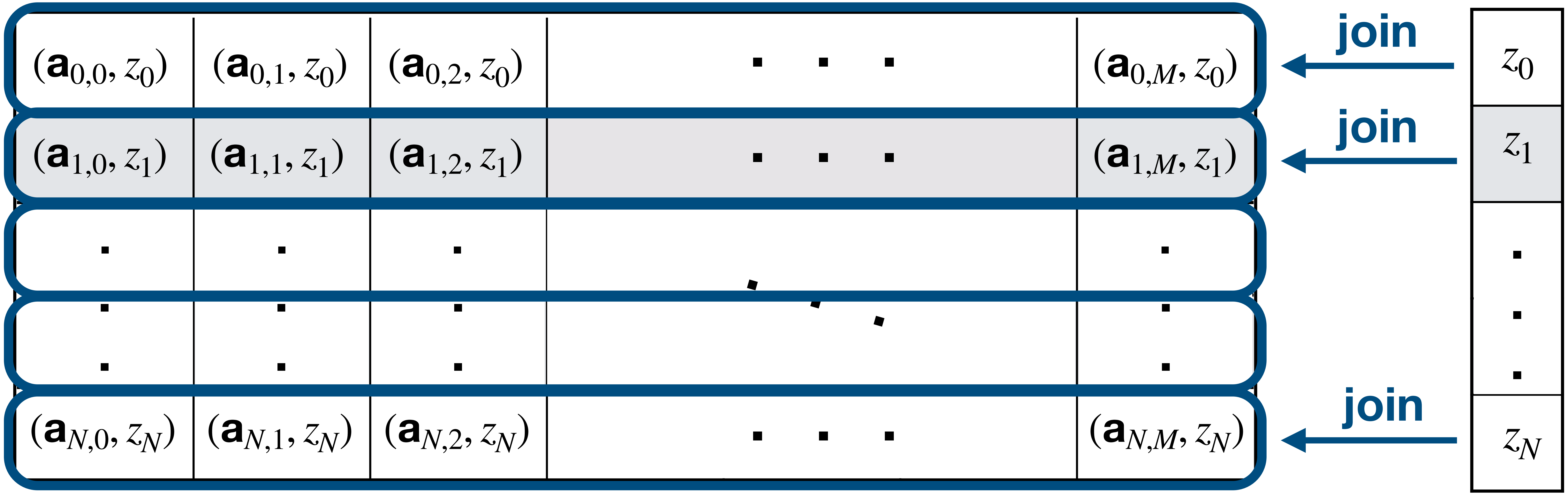
skewjoin

(Another common technique to address data skew. Source: *Tresata*)



`skewjoin` == Strawman

(Another common technique to address data skew. Source: *Tresata*)



skewjoin

(Another common technique to address data skew. Source: *Tresata*)

$(\mathbf{a}_{0,0}, z_0)$	$(\mathbf{a}_{0,1}, z_0)$	$(\mathbf{a}_{0,2}, z_0)$	$\cdots$	$(\mathbf{a}_{0,M}, z_0)$
$(\mathbf{a}_{1,0}, z_1)$	$(\mathbf{a}_{1,1}, z_1)$	$(\mathbf{a}_{1,2}, z_1)$	$\cdots$	$(\mathbf{a}_{1,M}, z_1)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$(\mathbf{a}_{N,0}, z_N)$	$(\mathbf{a}_{N,1}, z_N)$	$(\mathbf{a}_{N,2}, z_N)$	$\cdots$	$(\mathbf{a}_{N,M}, z_N)$

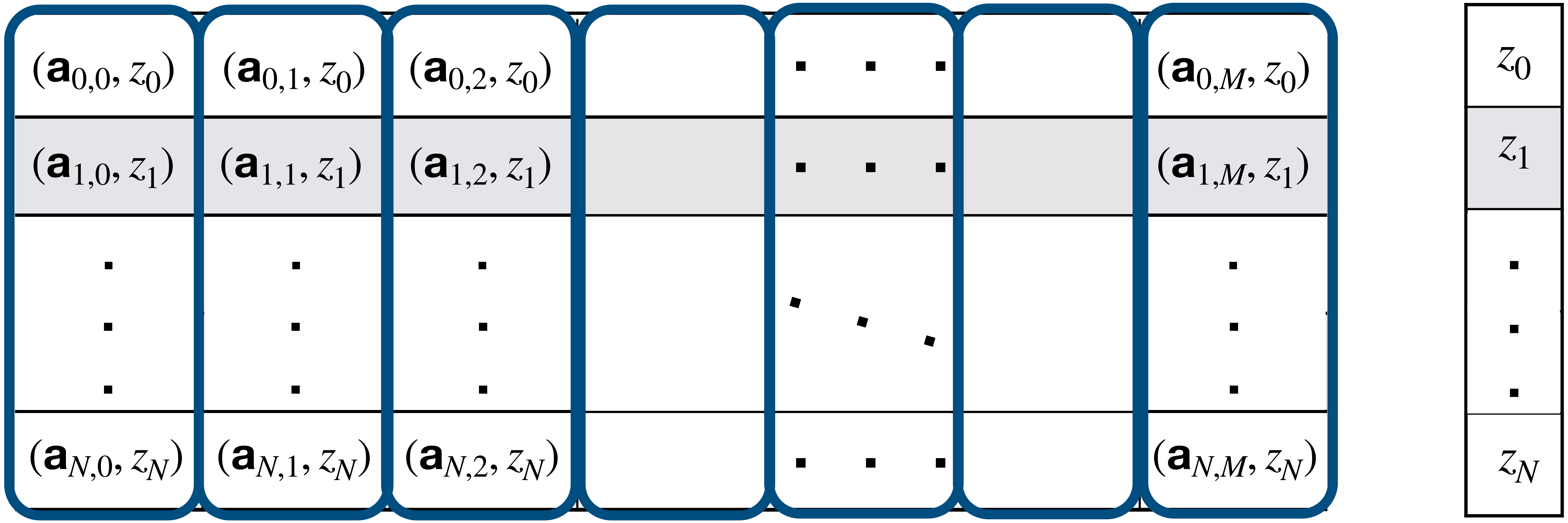
`skewjoin`

(Another common technique to address data skew. Source: *Tresata*)

$(\mathbf{a}_{0,0}, z_0)$	$(\mathbf{a}_{0,1}, z_0)$	$(\mathbf{a}_{0,2}, z_0)$		$\cdots$		$(\mathbf{a}_{0,M}, z_0)$
$(\mathbf{a}_{1,0}, z_1)$	$(\mathbf{a}_{1,1}, z_1)$	$(\mathbf{a}_{1,2}, z_1)$		$\cdots$		$(\mathbf{a}_{1,M}, z_1)$
$\vdots$	$\vdots$	$\vdots$		$\ddots$		$\vdots$
$(\mathbf{a}_{N,0}, z_N)$	$(\mathbf{a}_{N,1}, z_N)$	$(\mathbf{a}_{N,2}, z_N)$		$\cdots$		$(\mathbf{a}_{N,M}, z_N)$

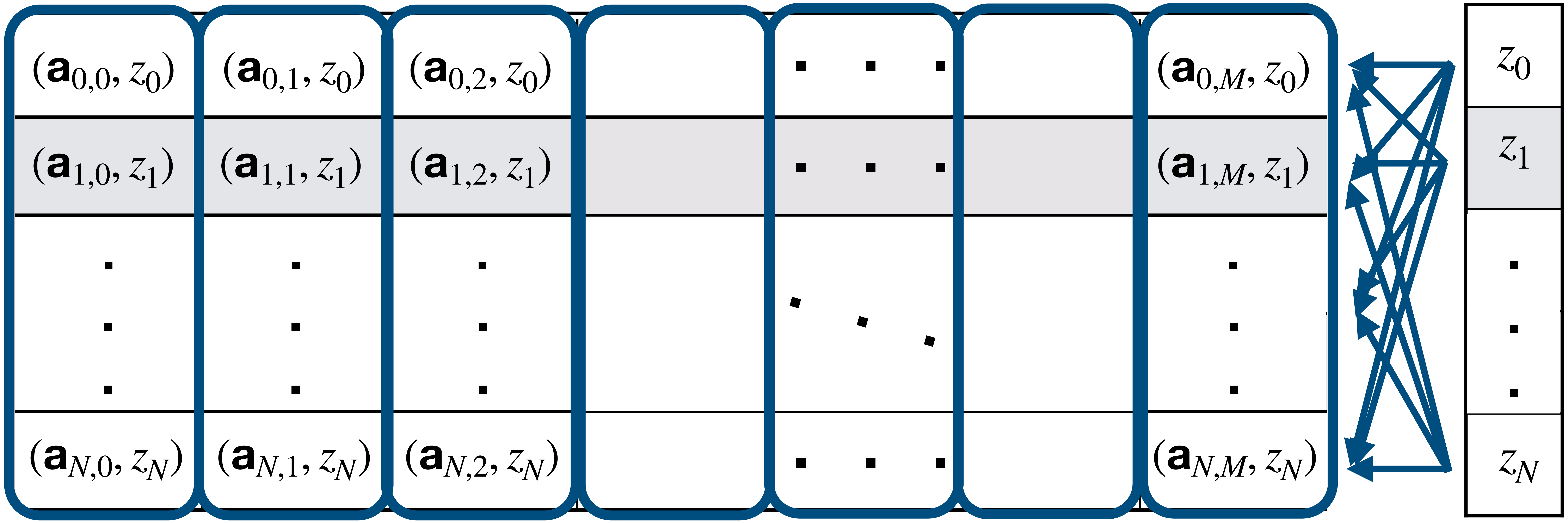
skewjoin

(Another common technique to address data skew. Source: *Tresata*)



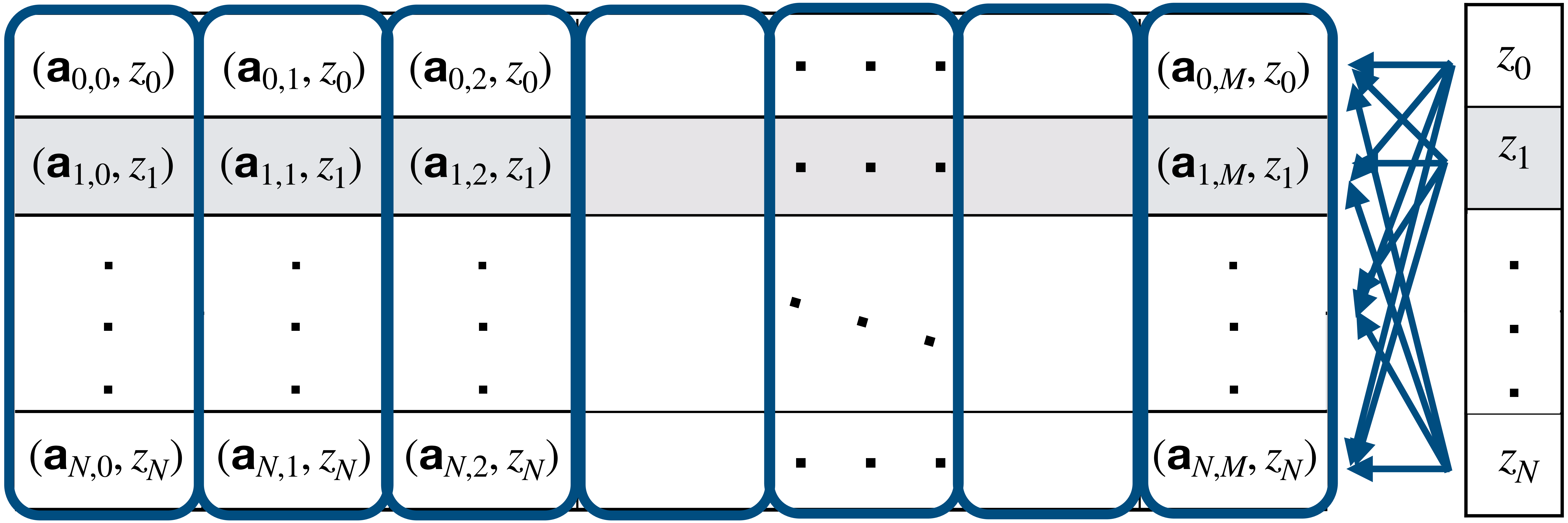
skewjoin

(Another common technique to address data skew. Source: *Tresata*)



`skewjoin` == `blockjoin`

(Another common technique to address data skew. Source: *Tresata*)



Tailored Approach

Tailored Approach

Isolate and ***transform*** the data
so that the computation is **distributed evenly.**

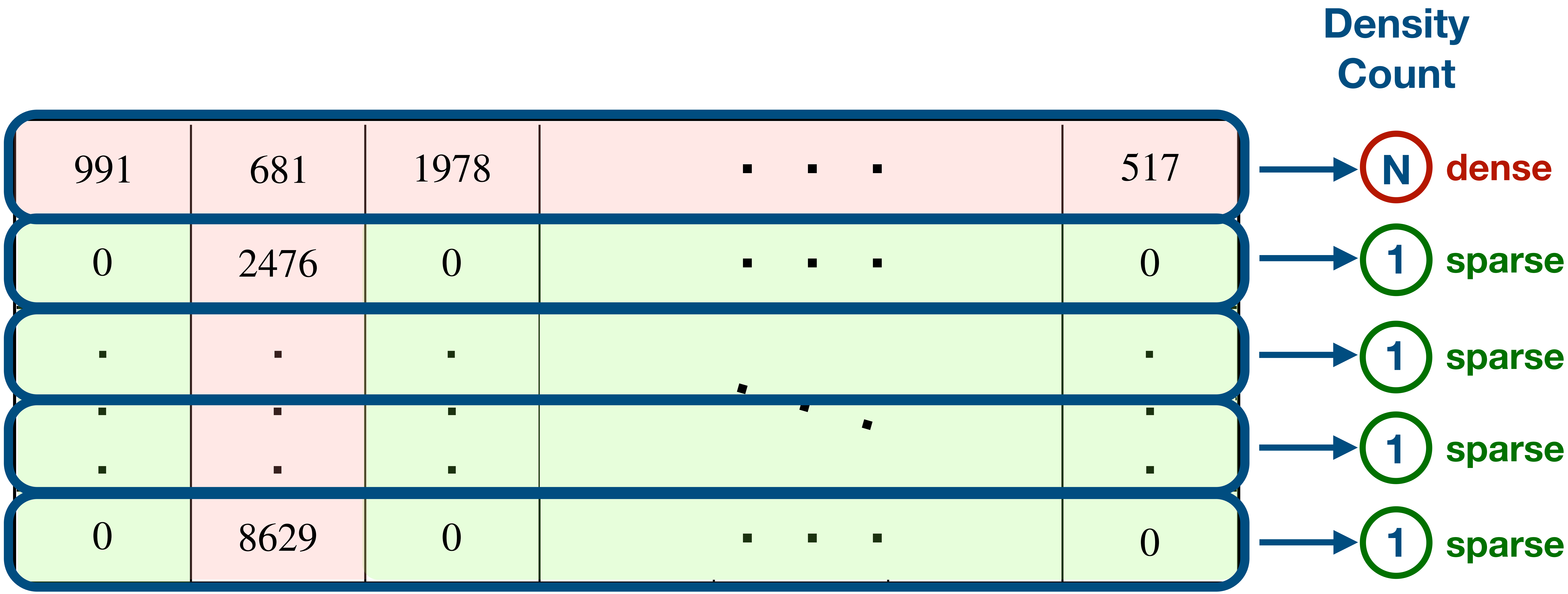
Tailored Approach — Part 1

Identify Dense Vectors

991	681	1978	. . .	517
0	2476	0	. . .	0
⋮	⋮	⋮	⋱	⋮
0	8629	0	. . .	0

Tailored Approach — Part 1

Identify Dense Vectors



Tailored Approach — Part 2

Employ a Hybrid Solution

Density Count

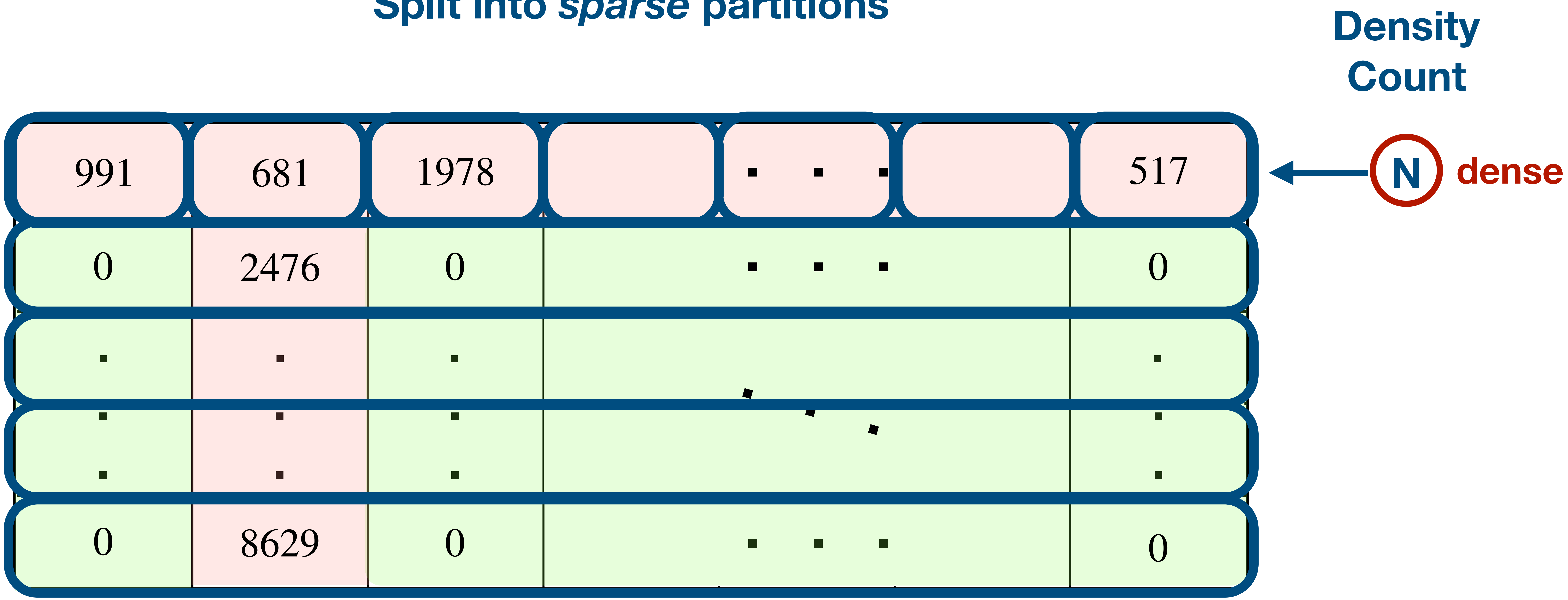
991	681	1978	. . .	517
0	2476	0	. . .	0
⋮	⋮	⋮	⋱	⋮
0	8629	0	. . .	0

N dense

Tailored Approach — Part 2

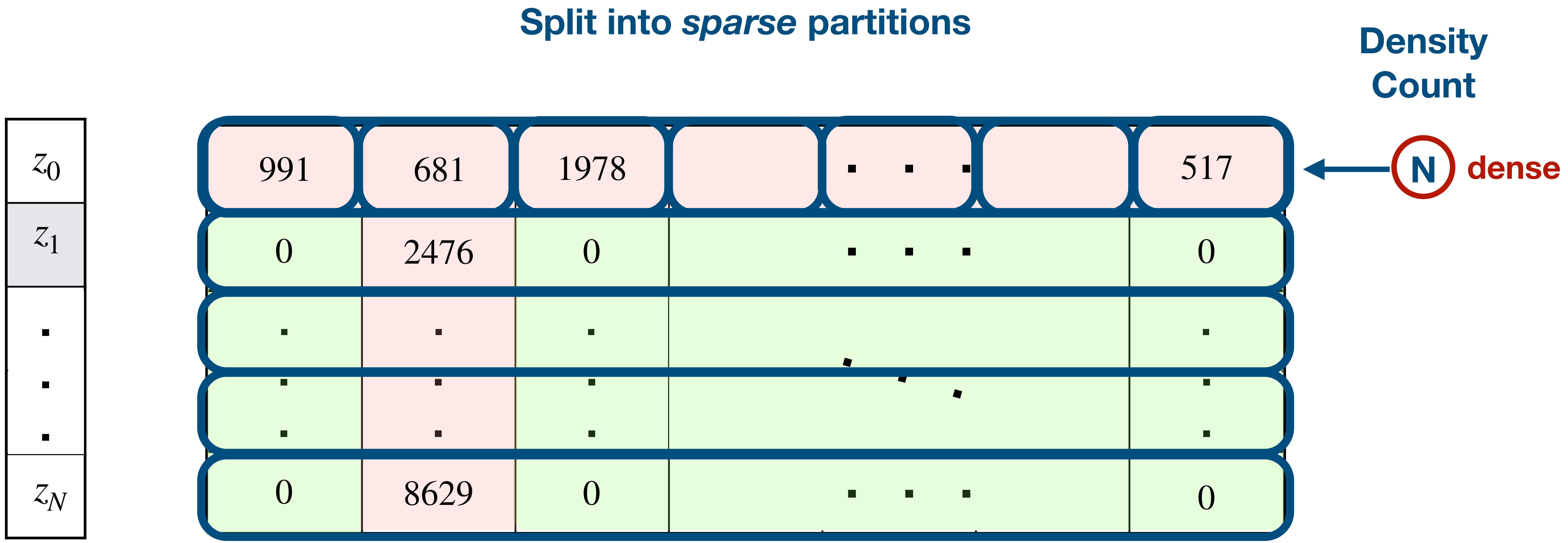
Employ a Hybrid Solution

Split into *sparse* partitions



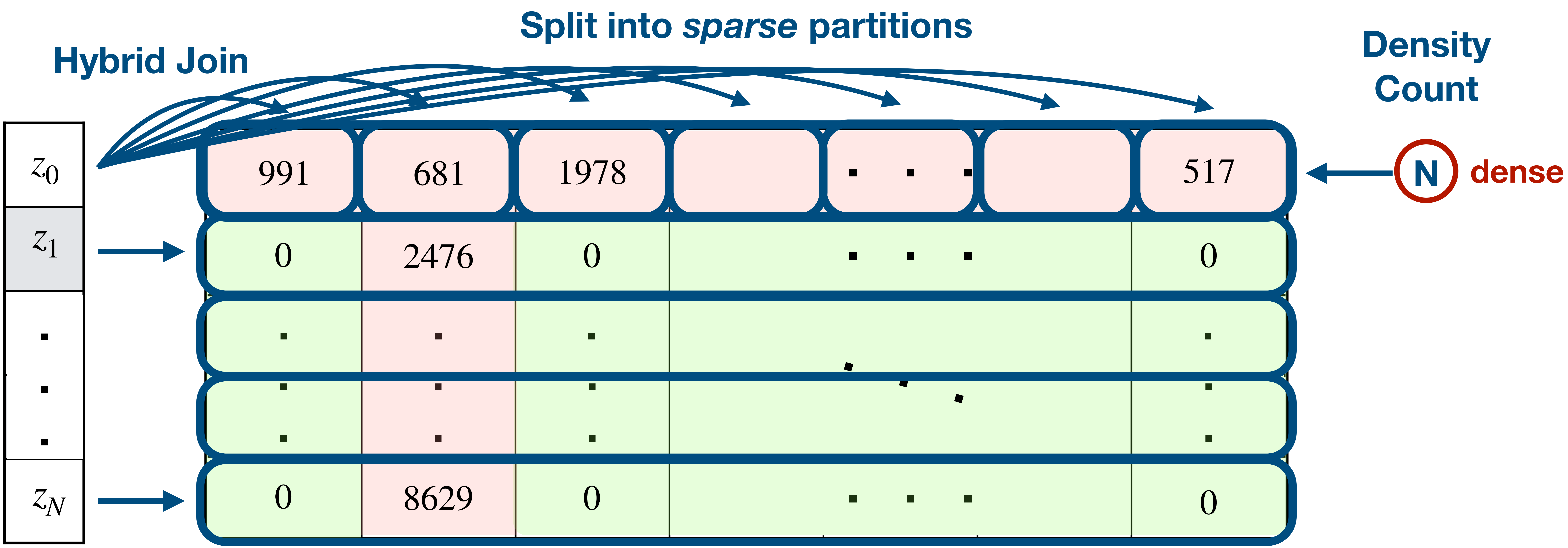
Tailored Approach — Part 2

Employ a Hybrid Solution



Tailored Approach — Part 2

Employ a Hybrid Solution



Tailored Approach — Part 2

Employ a Hybrid Solution

Each partition has just 1 nonzero computation

$(991, z_0)$	$(681, z_0)$	$(1978, z_0)$	$\cdots$	$(517, z_0)$
$(0, z_1)$	$(2476, z_1)$	$(0, z_1)$	$\cdots$	$(0, z_1)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$(0, z_N)$	$(8629, z_N)$	$(0, z_N)$	$\cdots$	$(0, z_N)$

Tailored Approach — Part 2

Employ a Hybrid Solution

Each partition has just 1 nonzero computation

$(991, z_0)$	$(681, z_0)$	$(1978, z_0)$		$\cdots$		$(517, z_0)$
$(0, z_1)$	$(2476, z_1)$	$(0, z_1)$		$\cdots$		$(0, z_1)$
$\vdots$	$\vdots$	$\vdots$		$\ddots$		$\vdots$
$(0, z_N)$	$(8629, z_N)$	$(0, z_N)$		$\cdots$		$(0, z_N)$

Tailored Approach — Part 2

Employ a Hybrid Solution

Each partition has just 1 nonzero computation

$991 \cdot z_0$	$681 \cdot z_0$	$1978 \cdot z_0$	$\cdots$	$\cdots$	$\cdots$	$517 \cdot z_0$
0	$2476 \cdot z_1$	0		$\cdots$		0
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
0	$8629 \cdot z_N$	0		$\cdots$		0

Implementation

Implementation

- Cluster-computing framework on **Apache Spark**

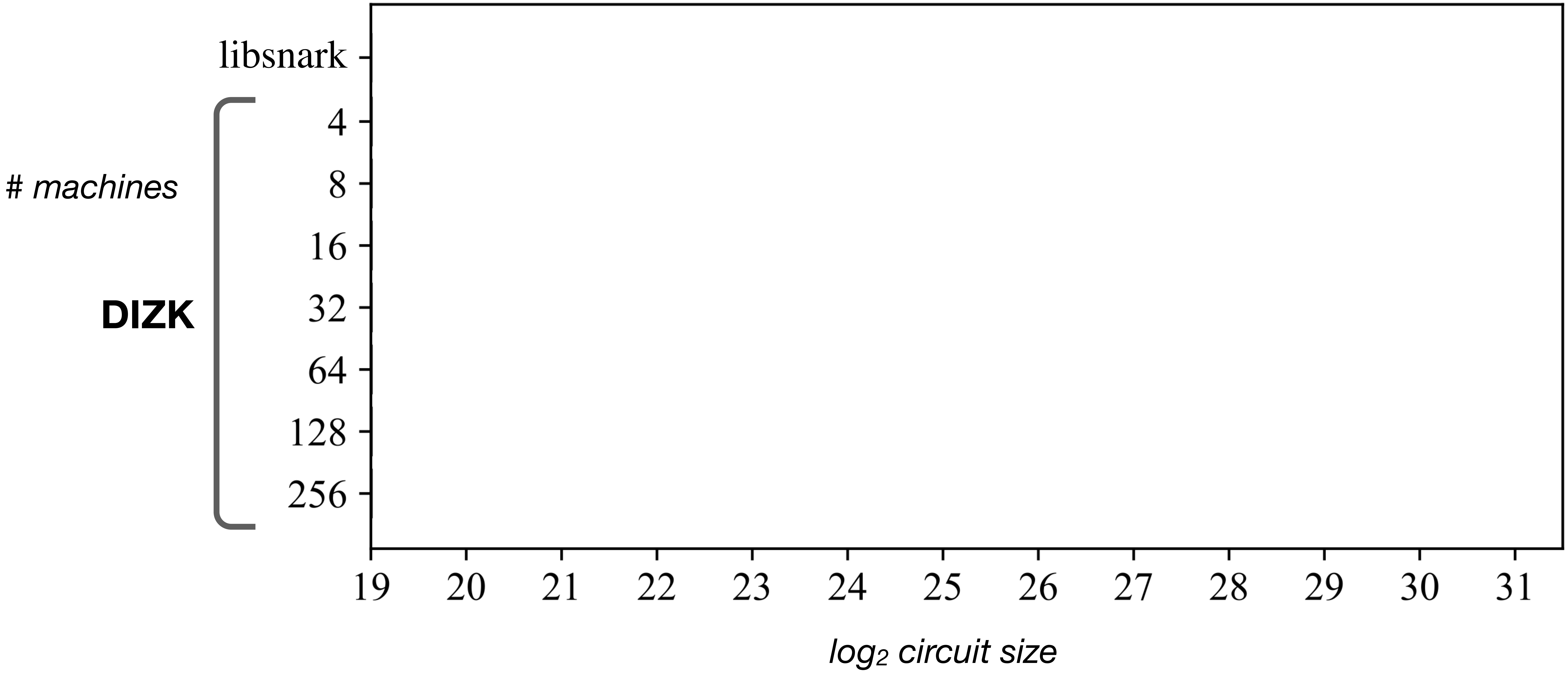
Implementation

- Cluster-computing framework on **Apache Spark**
- System written in **Java** (~10k lines of code)

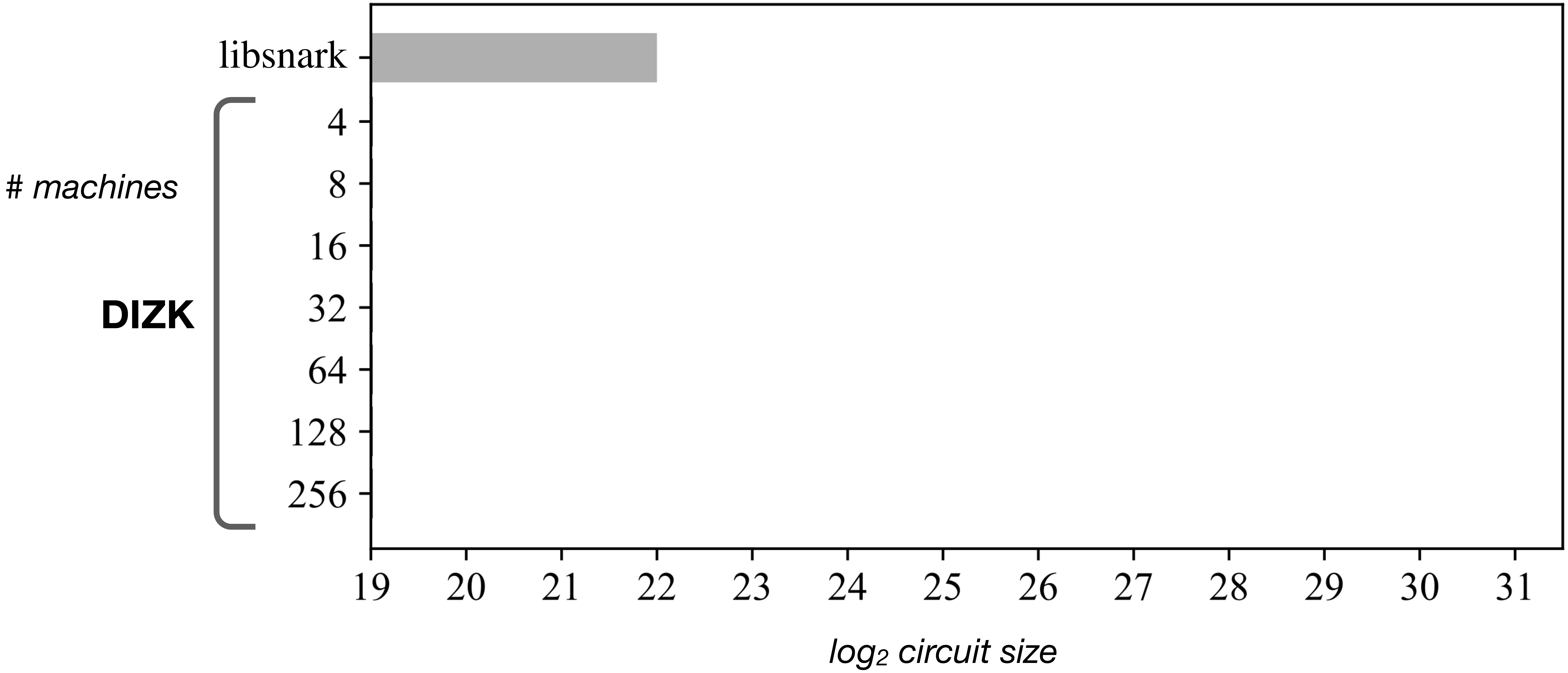
Implementation

- Cluster-computing framework on **Apache Spark**
- System written in **Java** (~10k lines of code)
- Experiments on **Amazon EC2**:
    - r3.8xlarge instances (32 vCPUs, 244 GiB of memory)

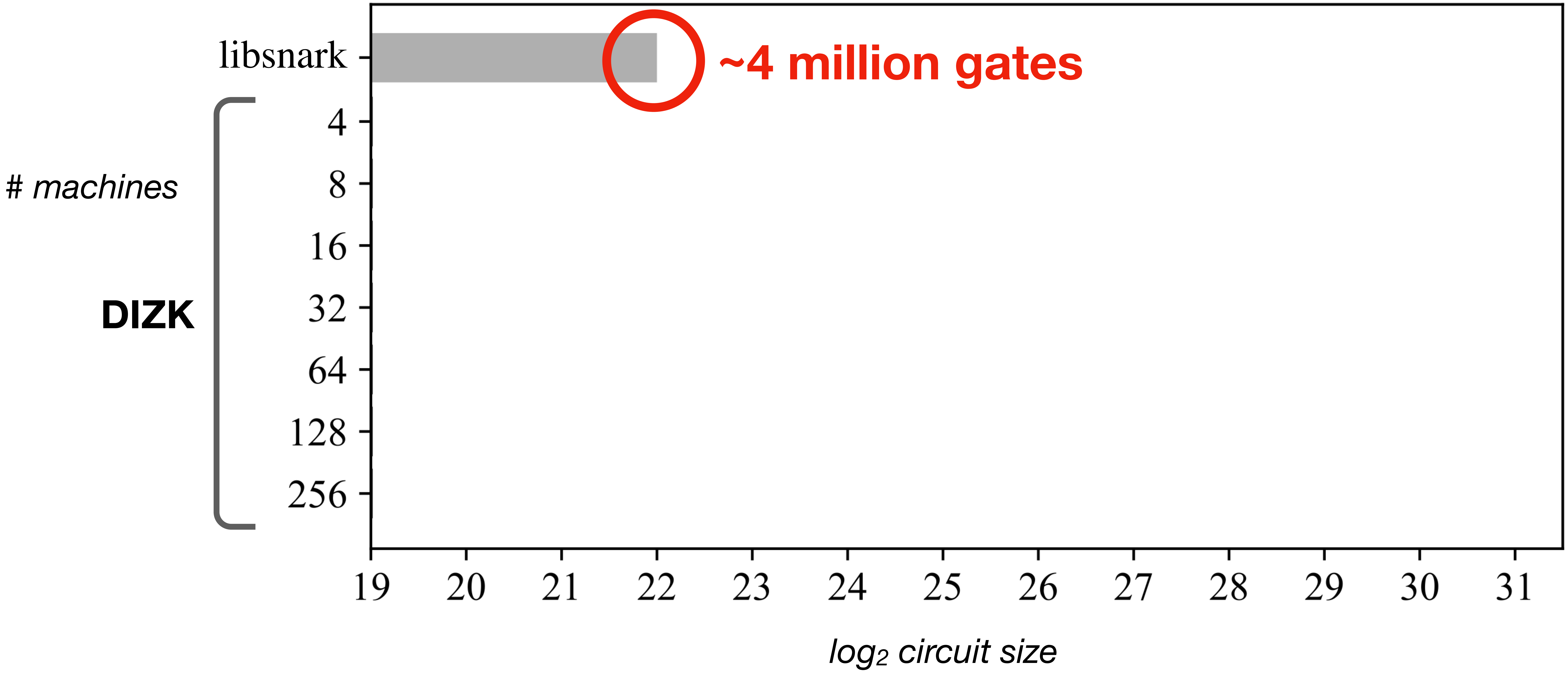
Largest Supported Circuit Size



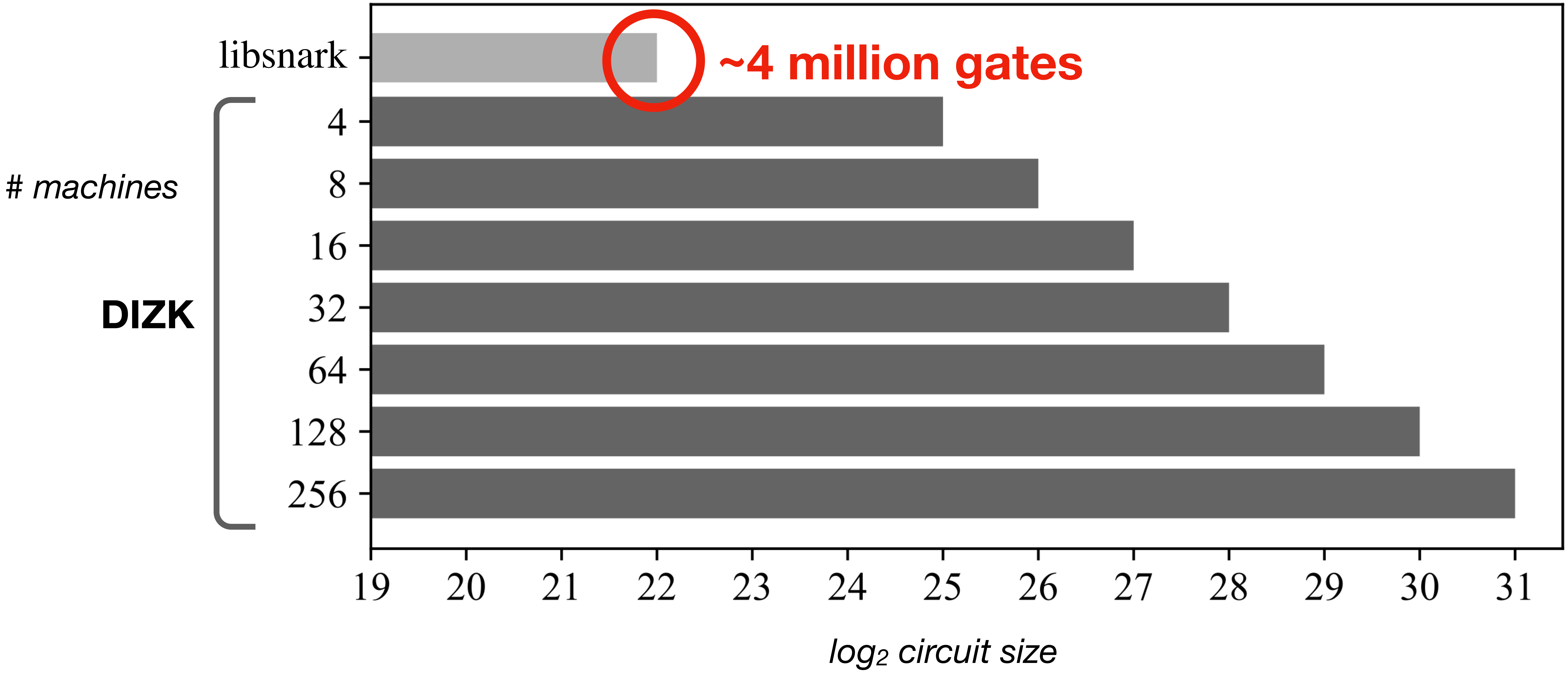
Largest Supported Circuit Size



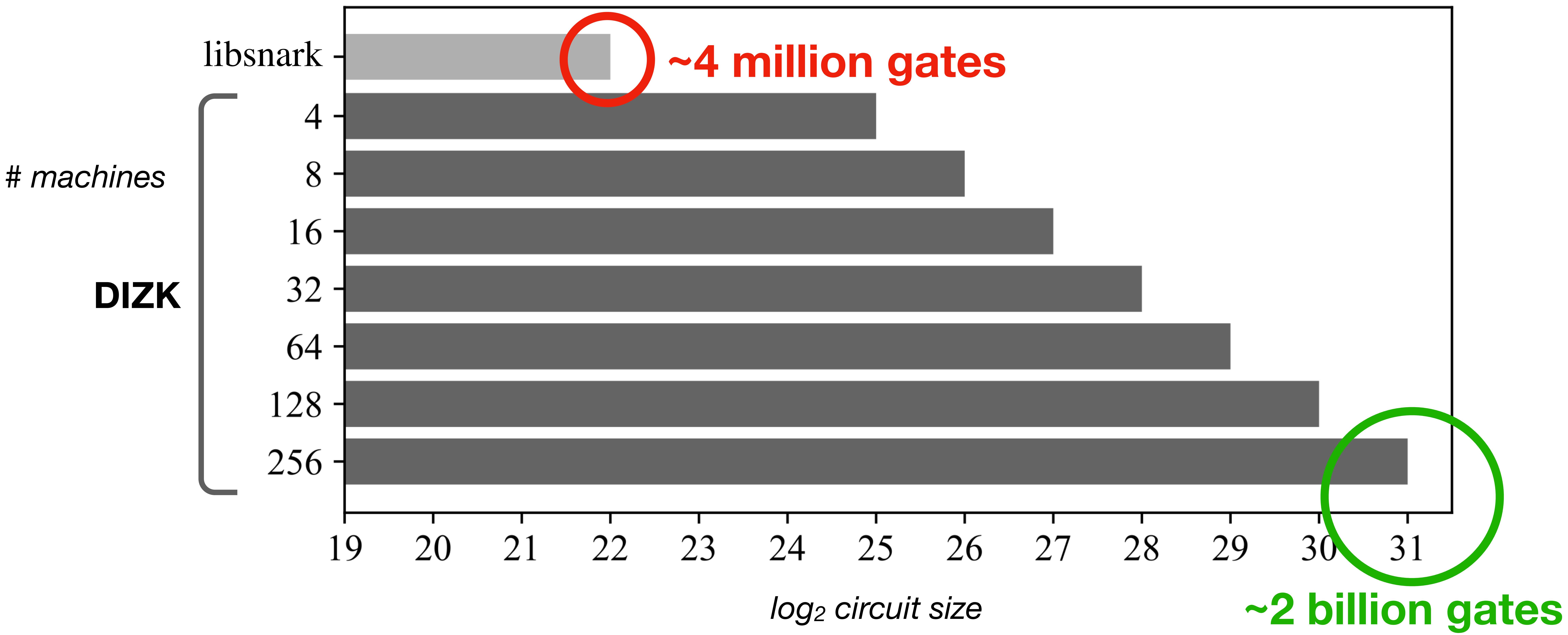
Largest Supported Circuit Size



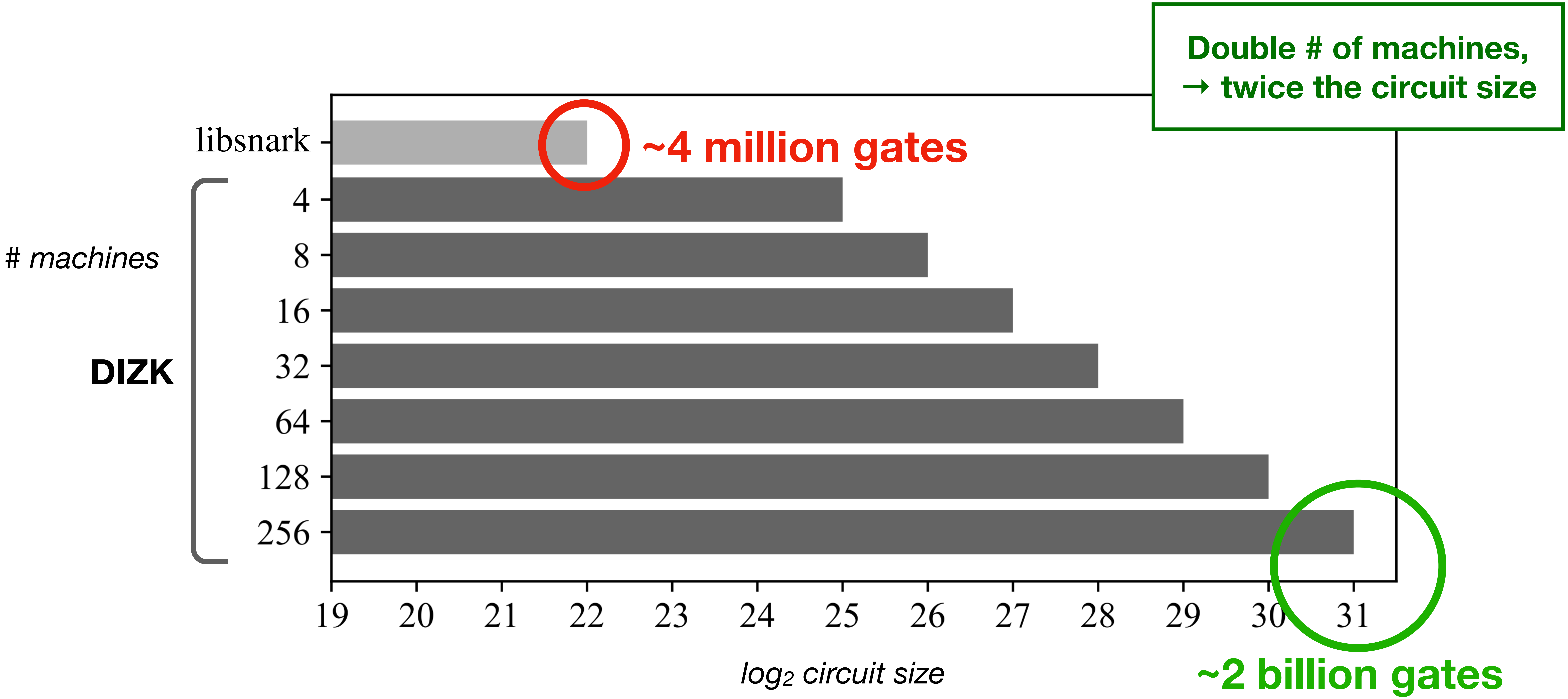
Largest Supported Circuit Size



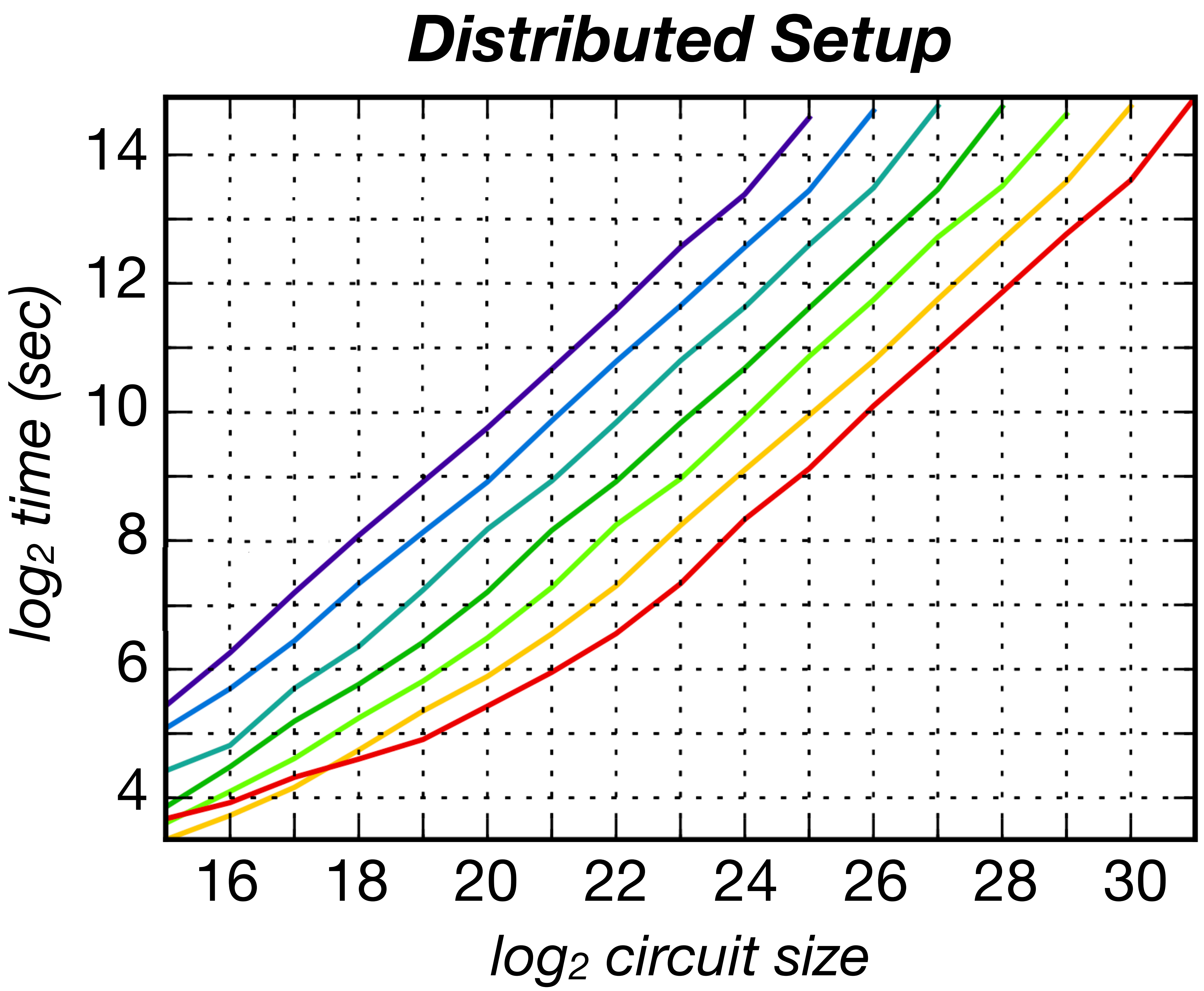
Largest Supported Circuit Size

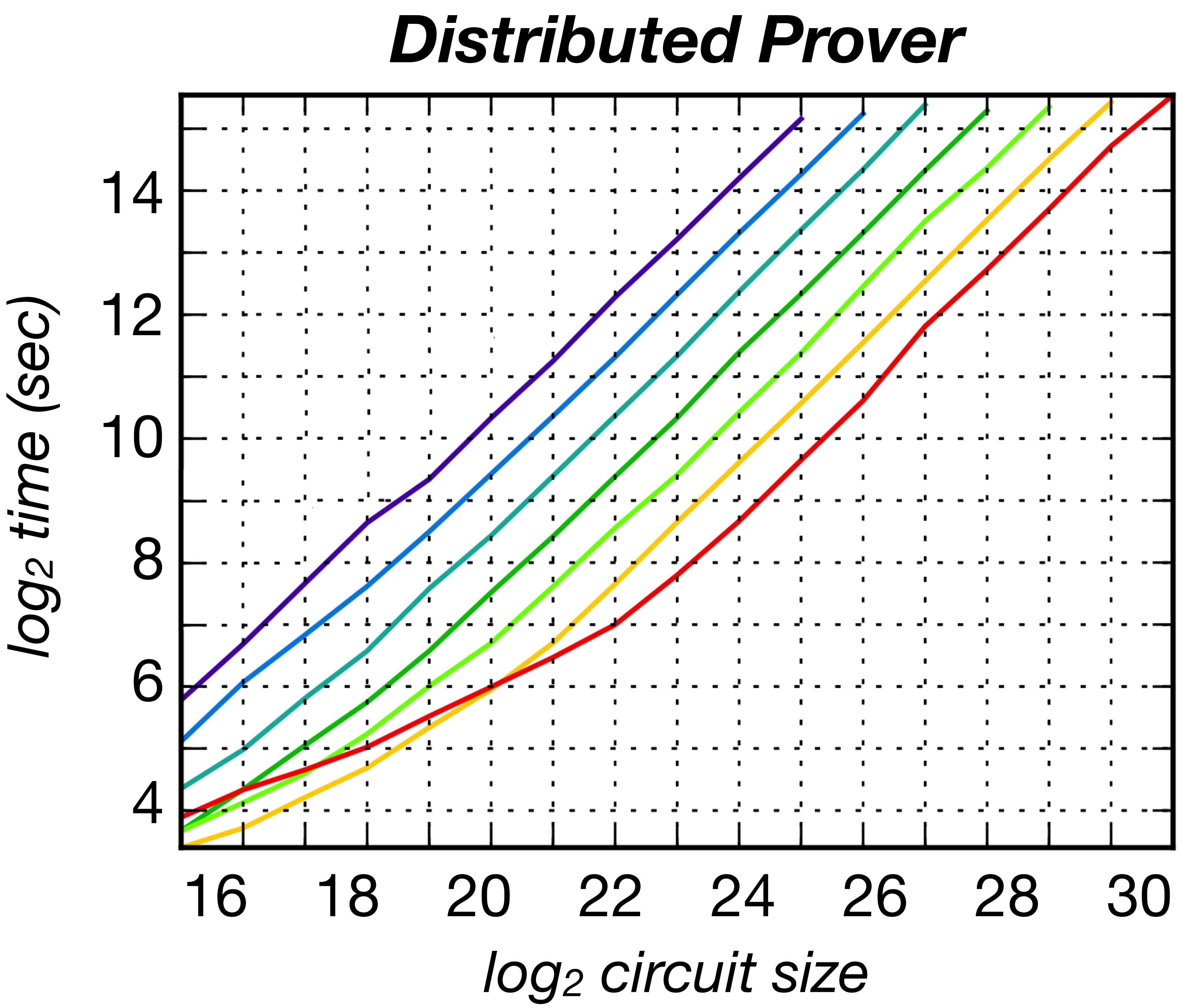


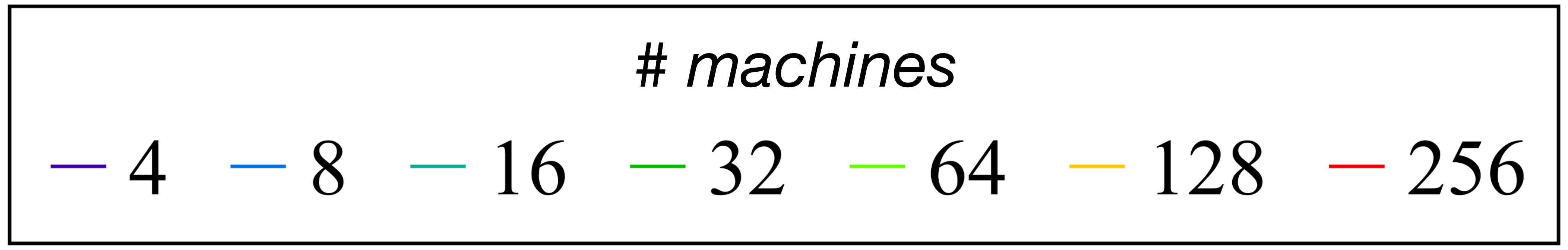
Largest Supported Circuit Size



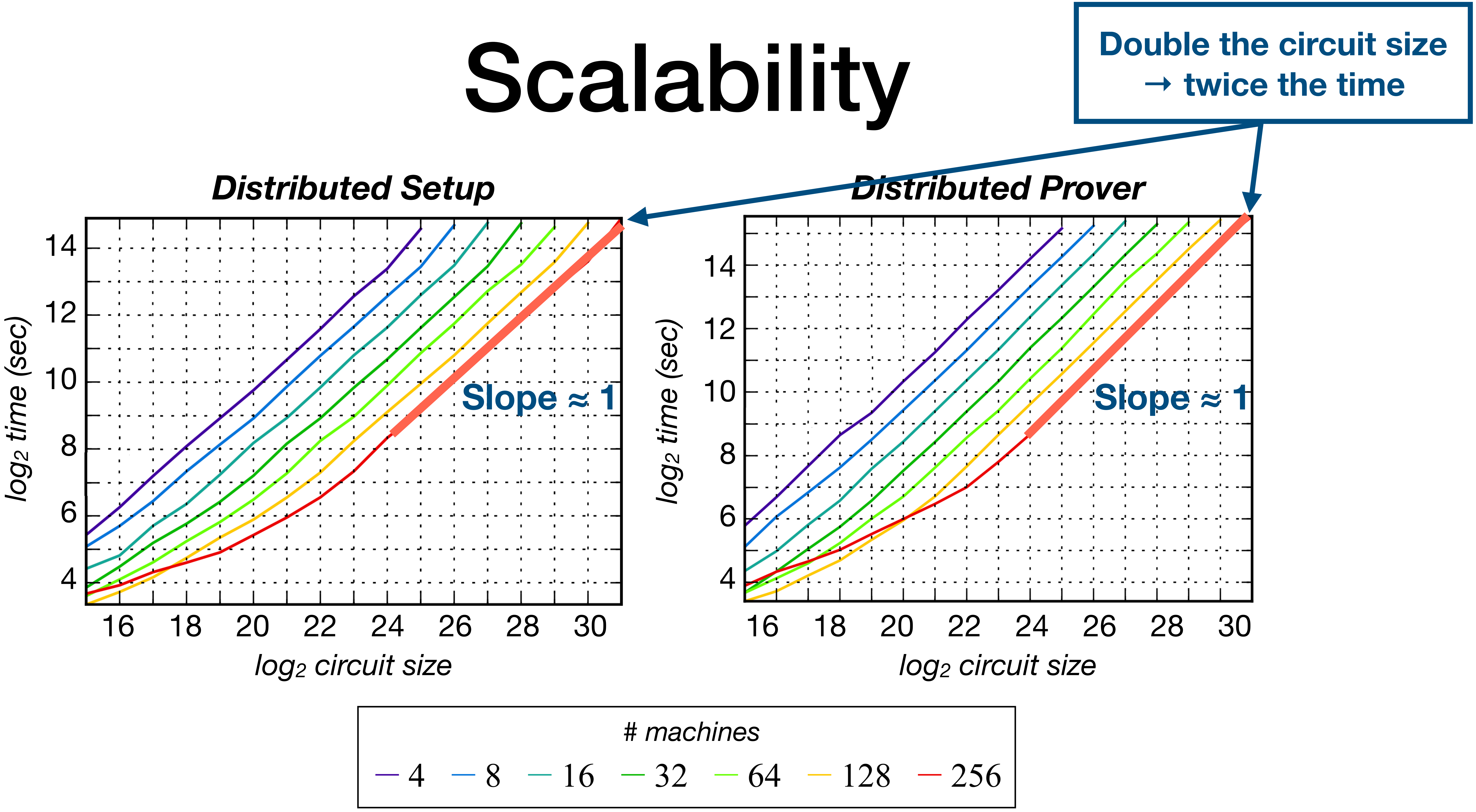
Scalability



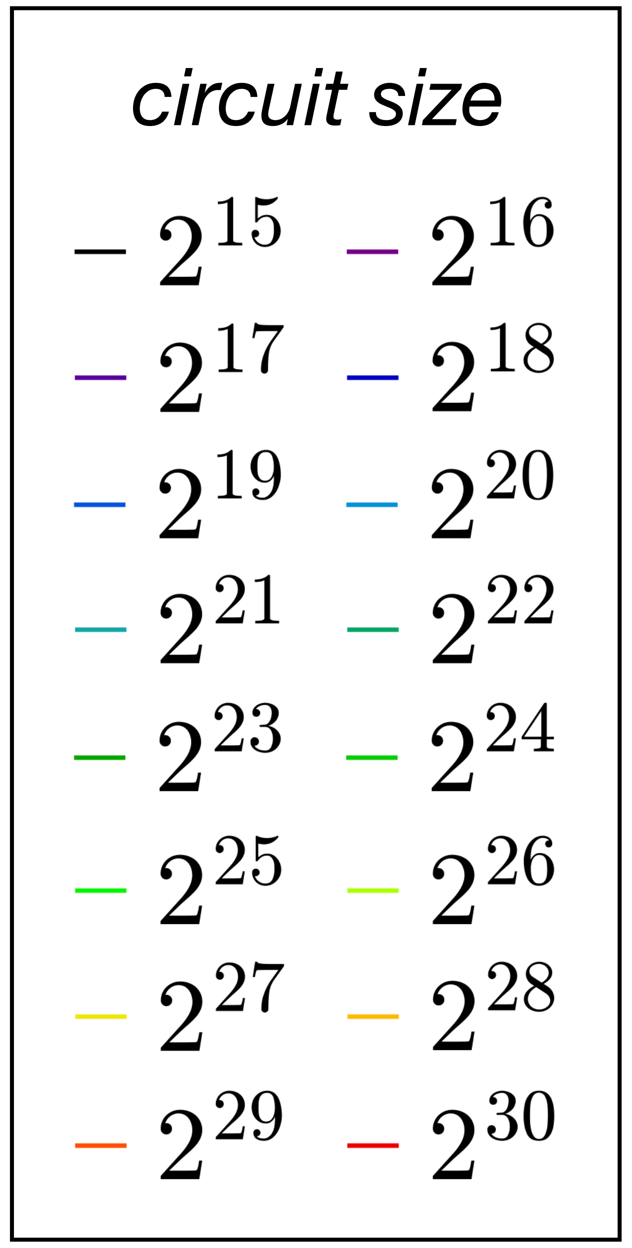


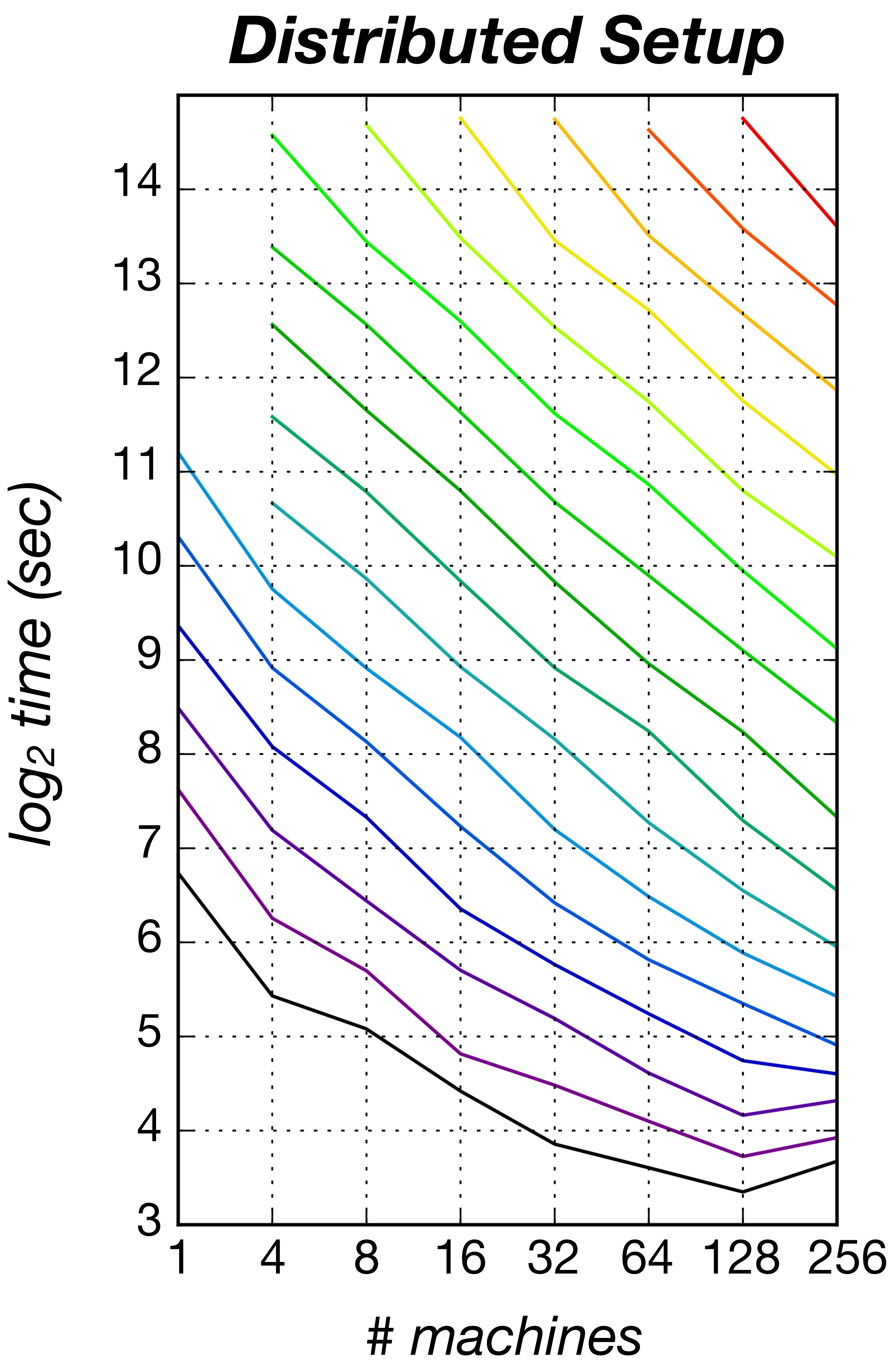


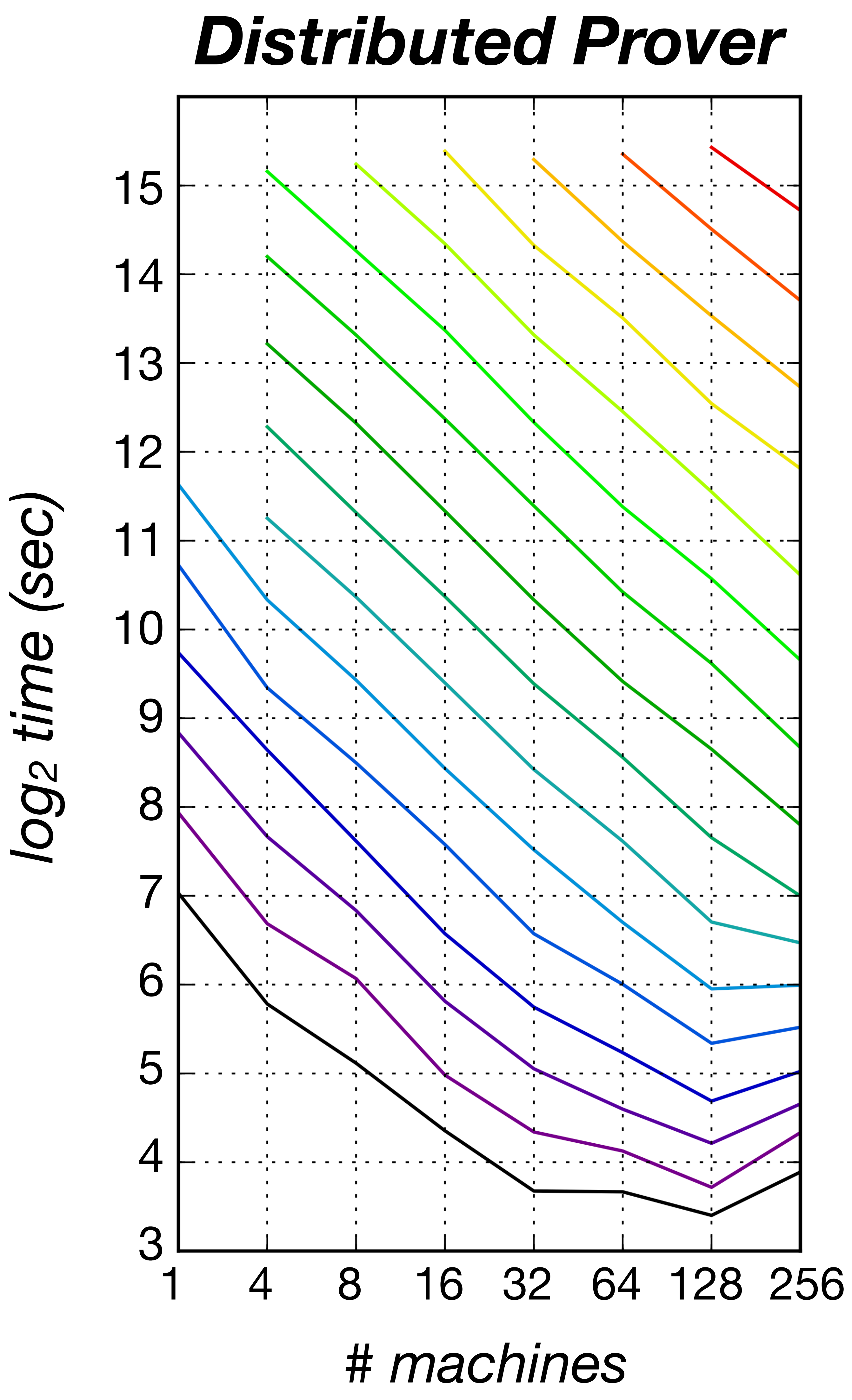
Scalability



Parallelism

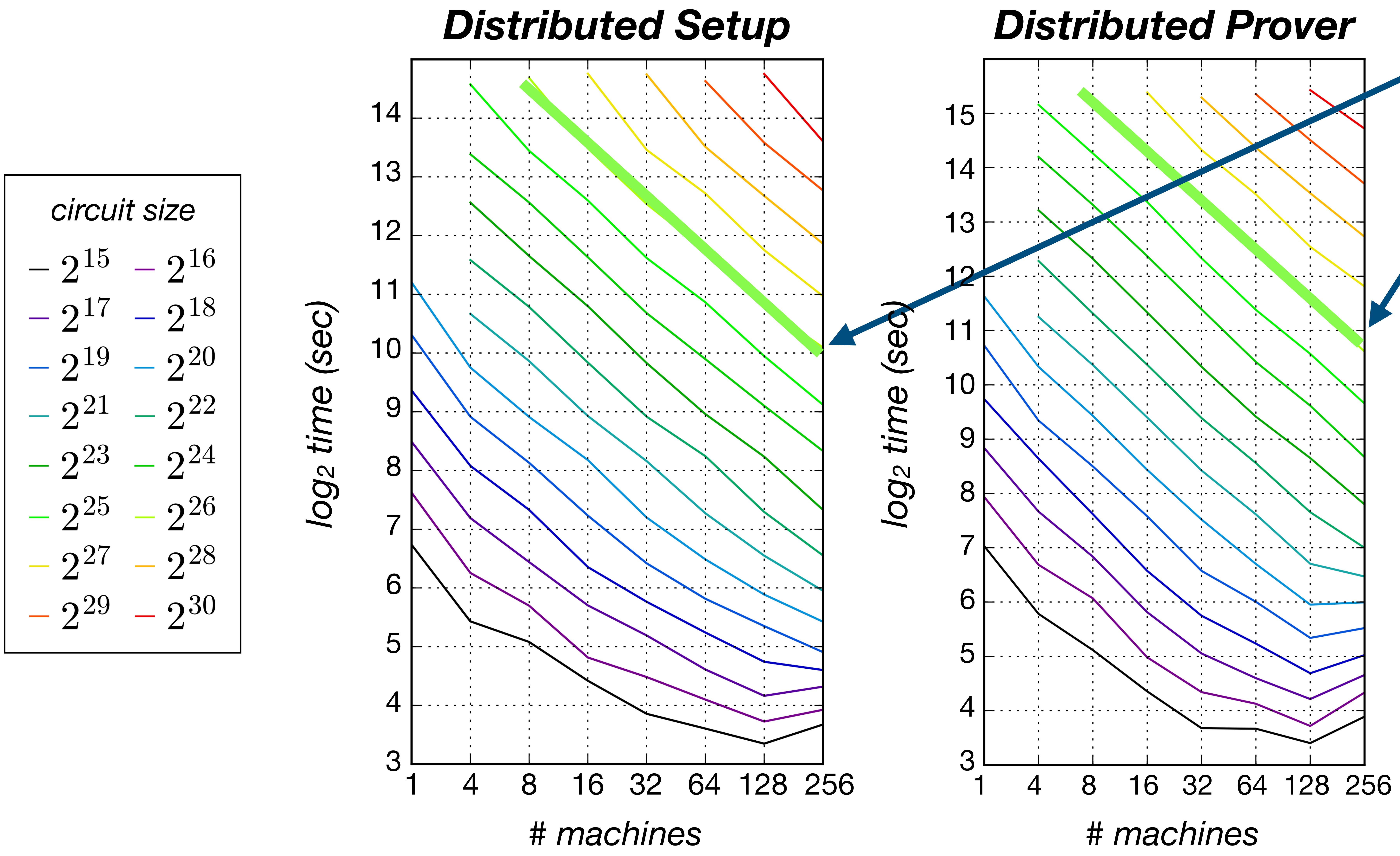






Parallelism

Double # of machines → twice as fast



Conclusion

Conclusion

	Prior zkSNARKs	DIZK
Maximum circuit size	**Millions of gates**	**Billions of gates**
Cost per gate	**1ms**	**10μs**

Conclusion

	Prior zkSNARKs	DIZK
Maximum circuit size	**Millions of gates**	**Billions of gates**
Cost per gate	**1ms**	**10μs**

- Full Paper on Crypto ePrint (**`https://eprint.iacr.org/2018/691`**)
- DIZK (**`dizk.org`**, open-source, MIT License)

Open Questions

Open Questions

Even Larger Circuits

What techniques will get us to **trillions of gates**, if any?
(Now, we would need ~100,000 machines in the best case scenario, i.e. too many)

Open Questions

Even Larger Circuits

What techniques will get us to **trillions of gates**, if any?
(Now, we would need ~100,000 machines in the best case scenario, i.e. too many)

Other Succinct ZKPs

How efficiently can **other succinct ZKPs** be distributed?
(STARKs, Bulletproofs, …)

Our techniques are likely an excellent starting point.