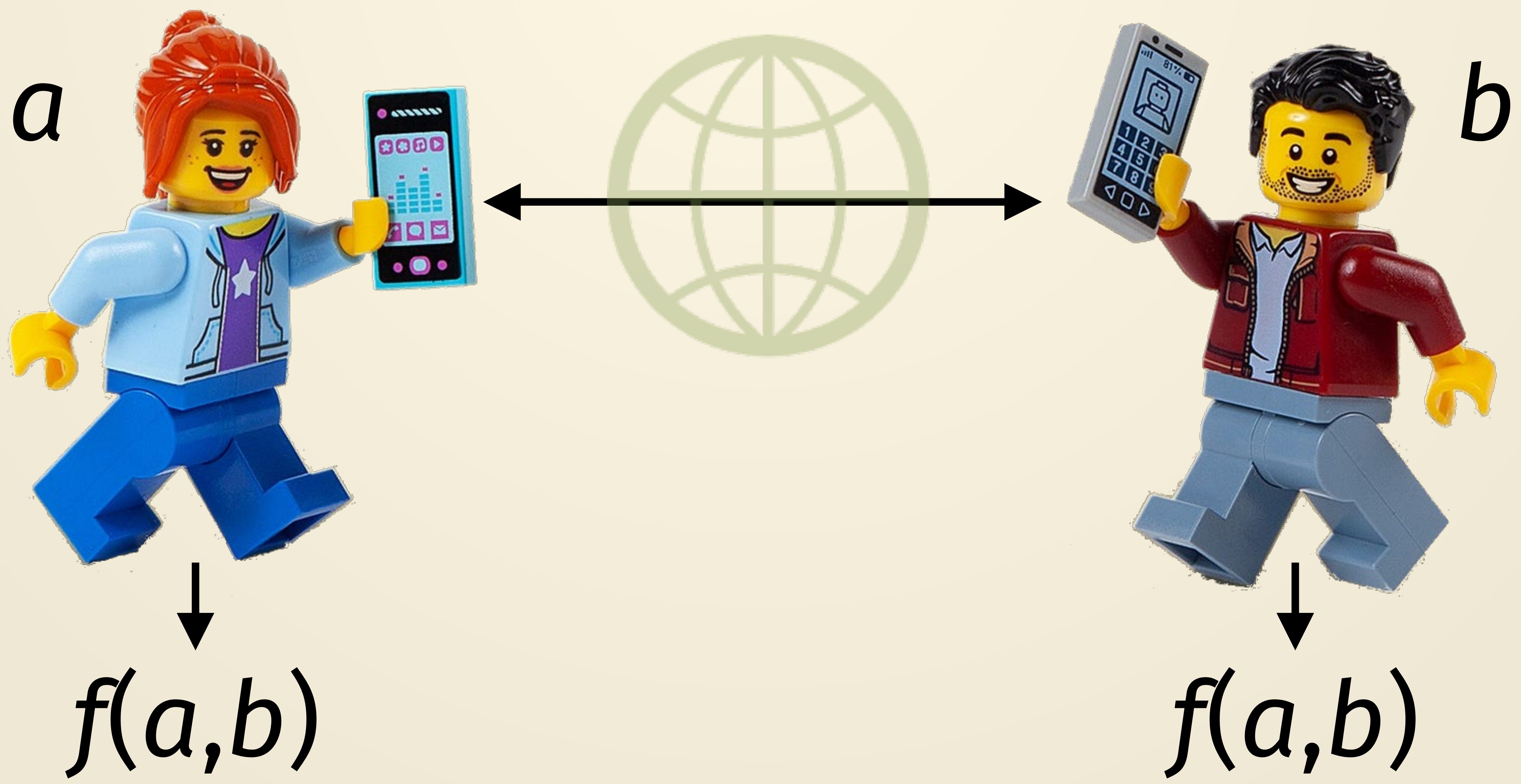
Course Announcements

- Office hours today will start at 11:40am
- Homework
  - Revisions to HW4 are due on Monday 3/20
  - Homework 5 will be released this Friday, and due next Friday 3/24
- Midterm tests will be graded and returned this weekend
- Project will be announced next week

Lecture 13:
Protecting Data in Use

Defining MPC (2022 U.S. Senate bill S.3952)

“Secure multi-party computation … enables *different participating entities* in possession of private sets of data to *link and aggregate their data sets* for the exclusive purpose of performing a finite number of pre-approved computations *without transferring or otherwise revealing any private data* to each other or anyone else.”



Objective of secure multi-party computation (MPC)

- Suppose m people have sensitive data $x_1, x_2, \ldots, x_m$
- Want to outsource this data to multiple compute parties $P_1, P_2, \ldots, P_n$
- Parties engage in computing a publicly-known function f

$y = f($ 

$,$ 

$, \ldots ,$ 

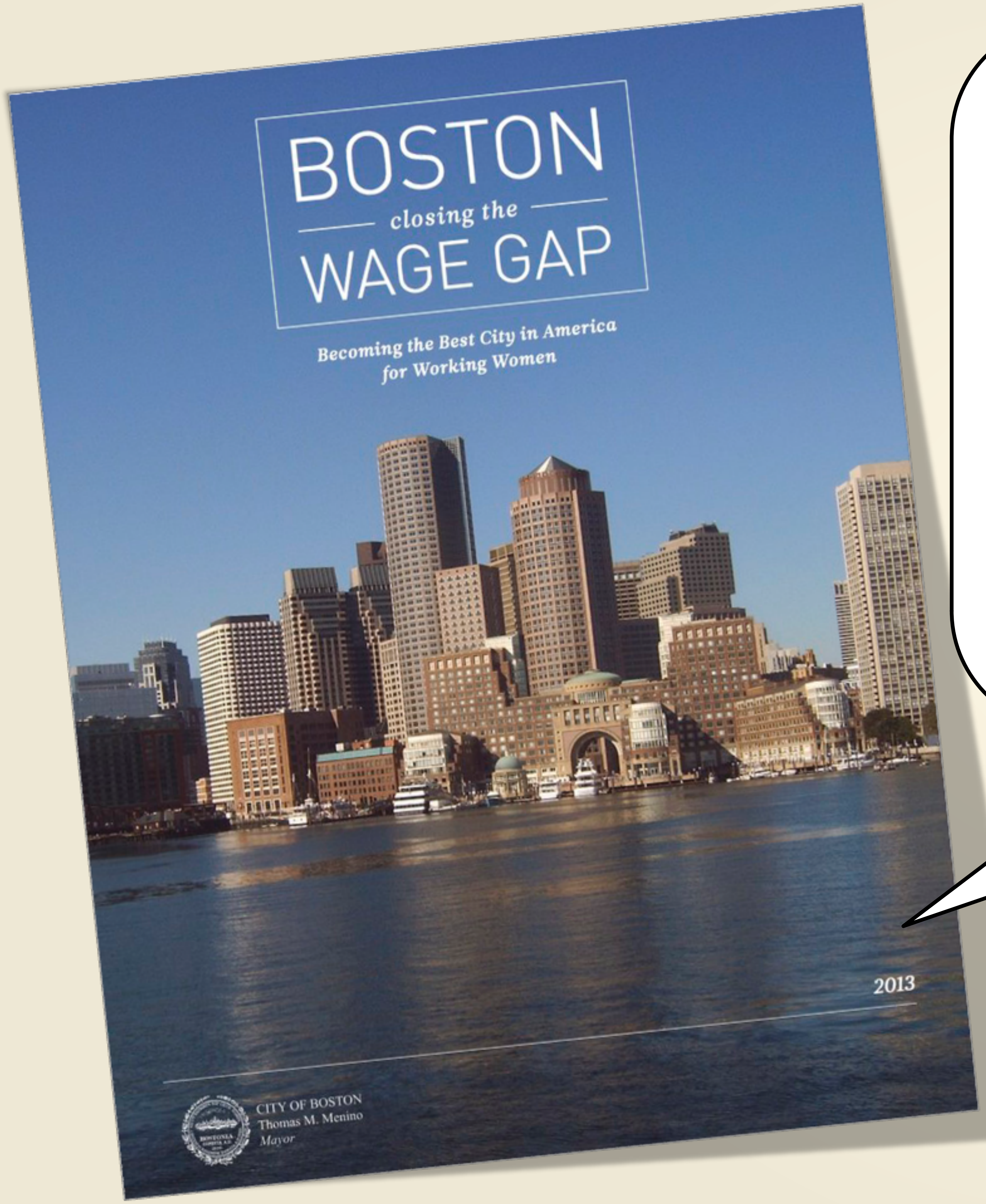
$)$

- Want to ensure: nothing is revealed about the inputs beyond what can be inferred from the output y (note: for some f, inference is bad!)

Computing in the presence of an adversary

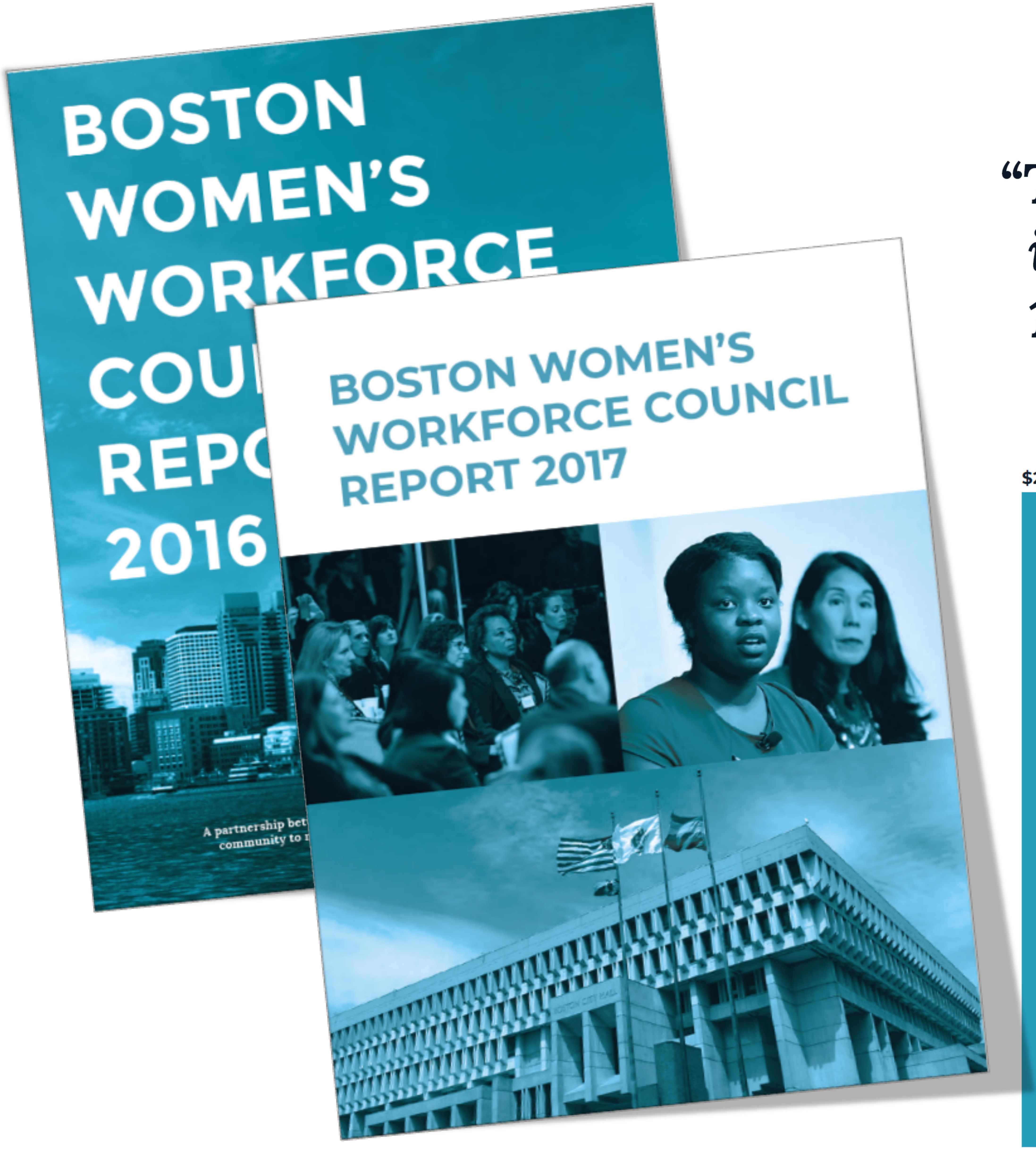
- Our concern is that up to t of the n parties are adversarial
- We will consider 3 kinds of security guarantees to enforce
  - Semi-honest security: withstands an adversary who follows the protocol but is trying to learn data (i.e., break confidentiality)
  - Malicious security: withstands an adversary who also might deviate from the protocol to learn data or alter the results of the computation (break integrity)
  - Robustness: withstands an adversary who also might quit participation (break availability), and will reach agreement on the result of the computation anyway
    - (This is similar to “agreement” in the setting of asynchronous protocols)

13.1 An Example

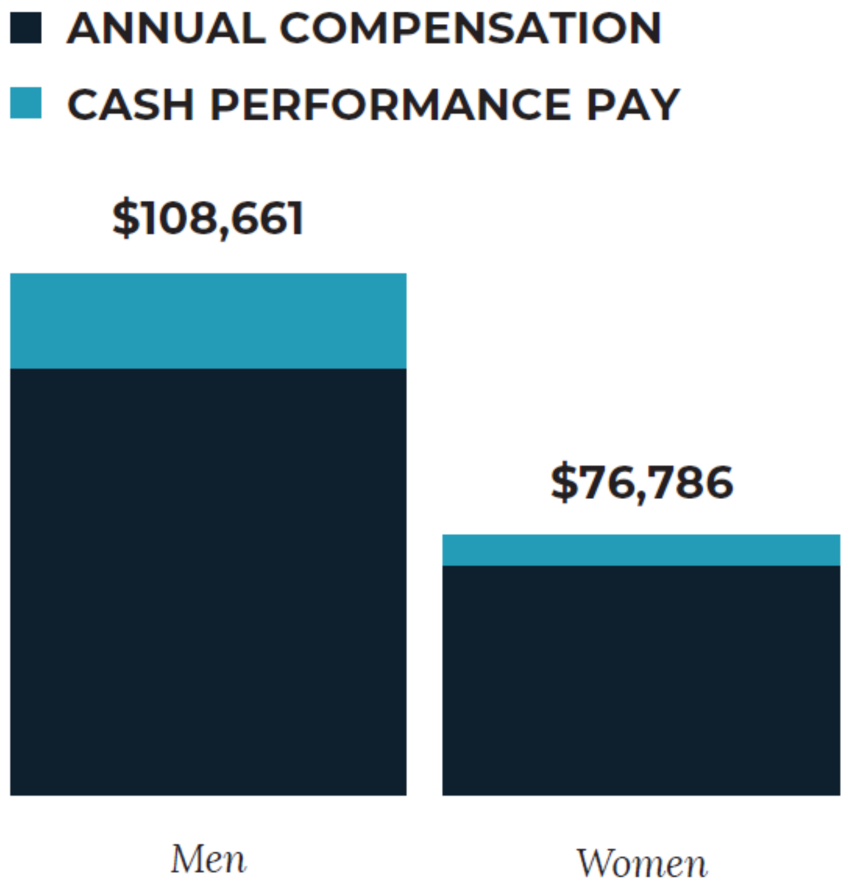


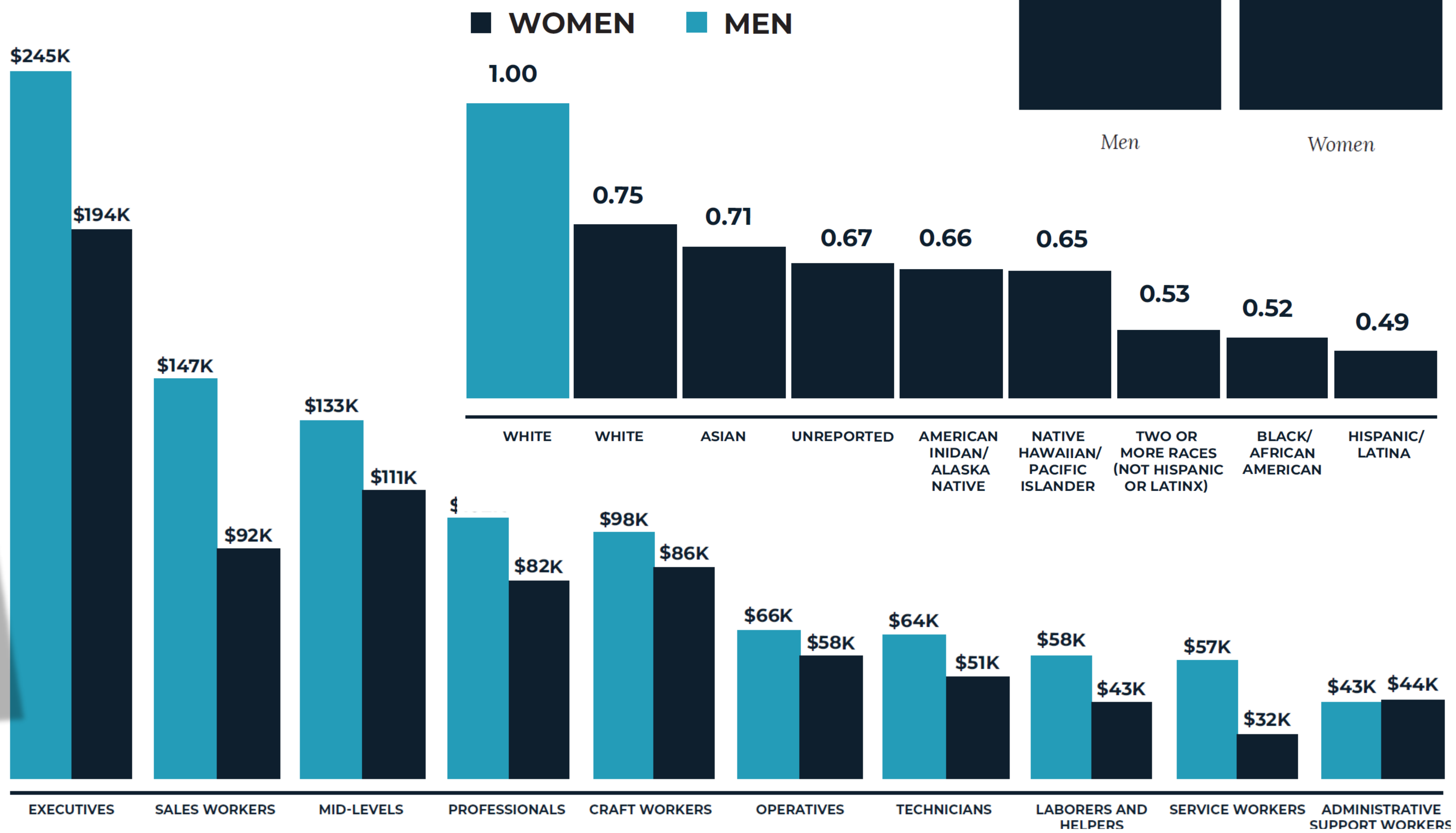
Goal 3: *Evaluating Success*

Employers agree to … contribute data to a report compiled by a third party on the Compact's success to date. Employer-level data would not be identified in the report.



"This year's data submission included 166,705 employees from 114 Compact-signing companies.





Boston Women's Workforce Co ×
Secure | https://100talent.org

Boston Women's Workforce Council

100% Talent Data Submission

BOSTON UNIVERSITY

Number Of Employees

	Hispanic or Latinx		White		Black/African American		Native Hawaiian or Pacific Islander		Asian		American Indian/Alaska Native		Two or More Races (Not Hispanic or Latinx)		Unreported	
	Female	Male	Female	Male	Female	Male	Female	Male	Female	Male	Female	Male	Female	Male	Female	Male
Executive/Senior Level Officials and Managers																
First/Mid-Level Officials and Managers																
Professionals																
Technicians																
Sales Workers																
Administrative Support Workers																
Craft Workers																
Operatives																
Laborers and Helpers																
Service Workers																

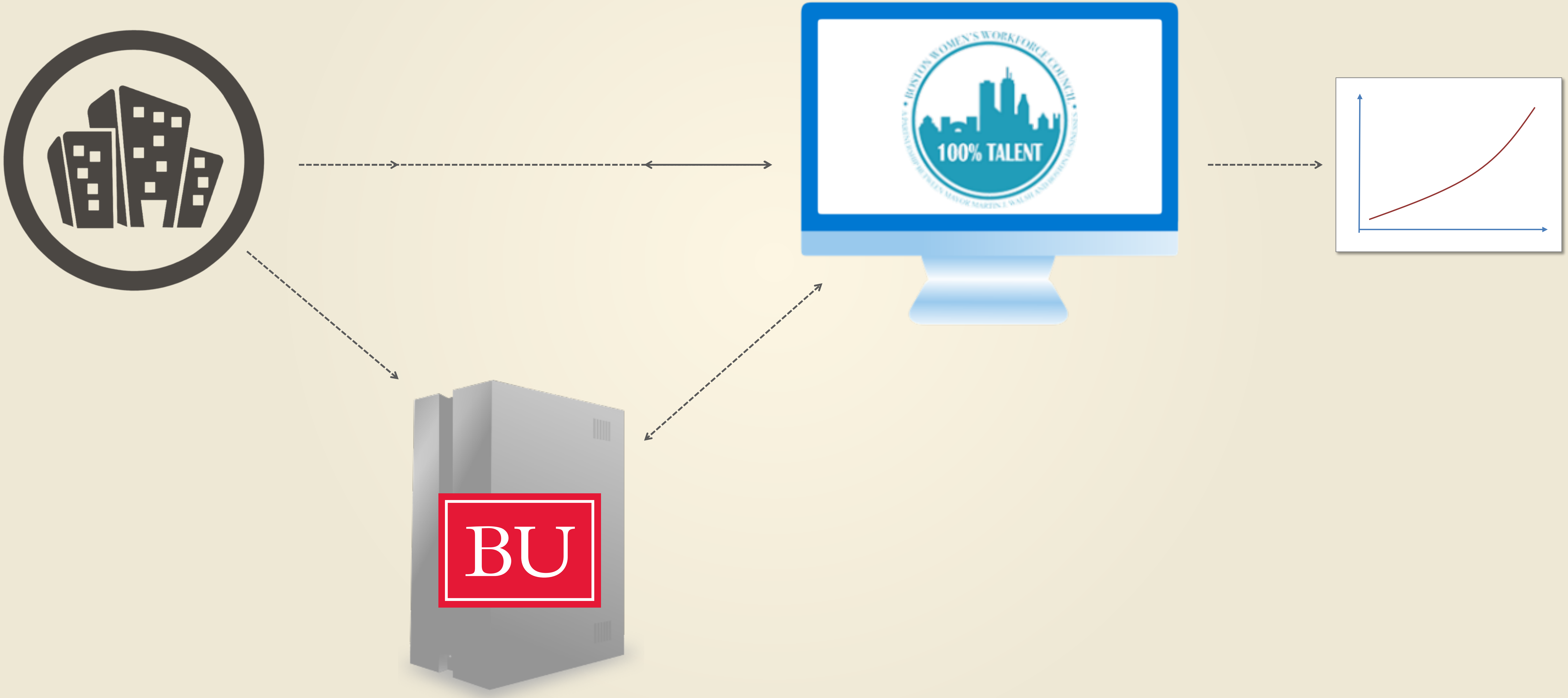
Trust spectrum

Trust us

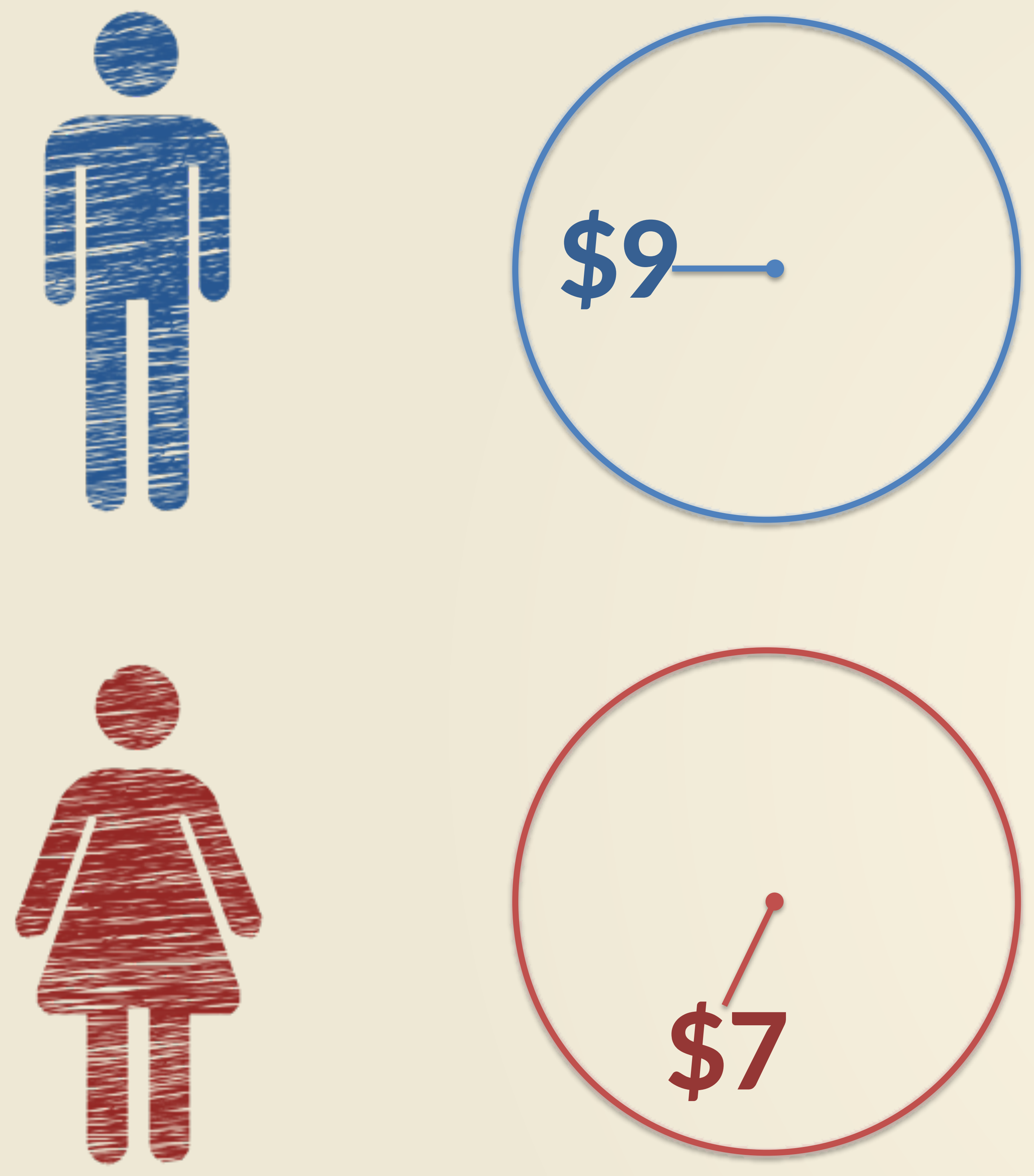
Trust anyone

Trust no one

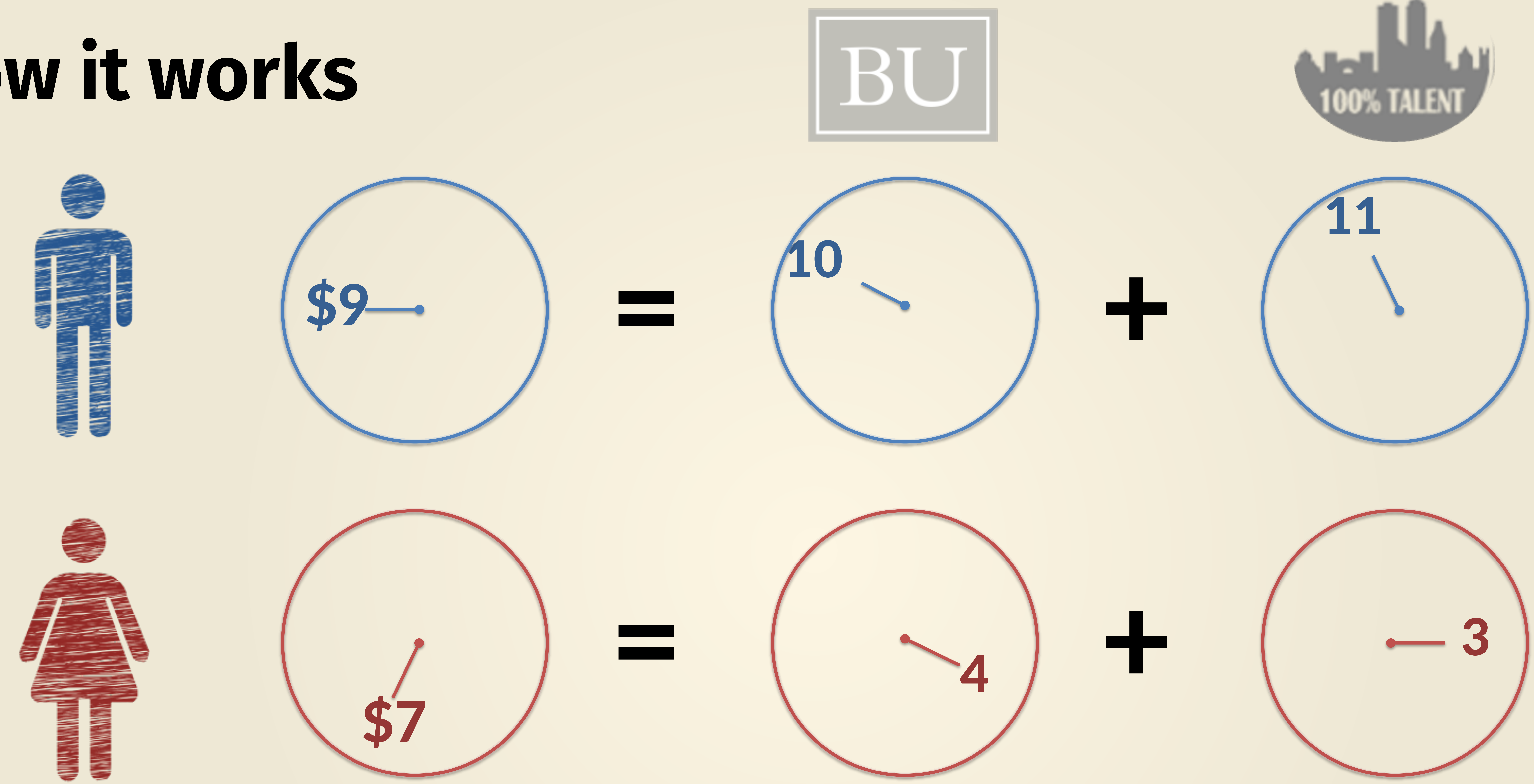
Workflow



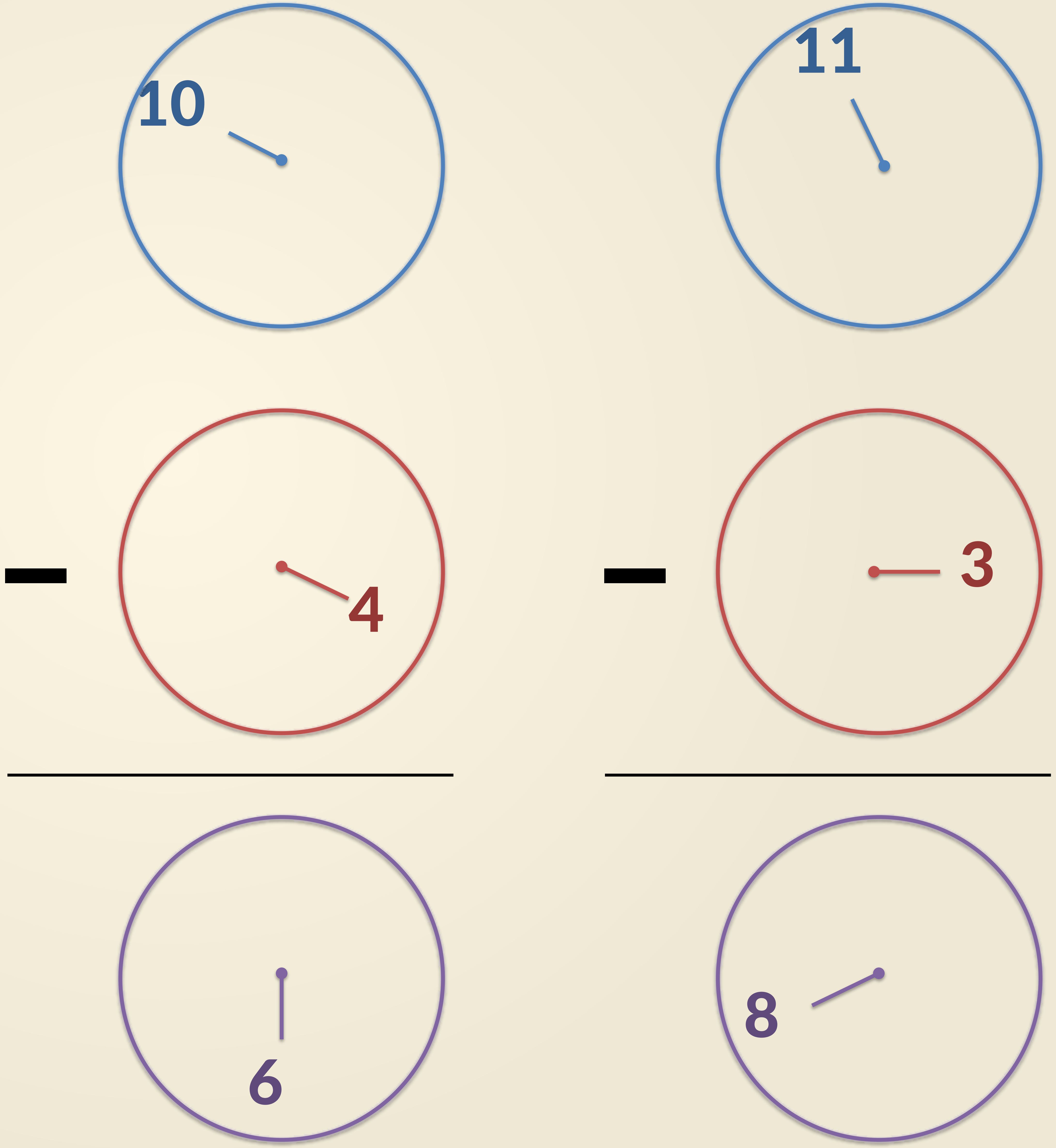
How it works



How it works

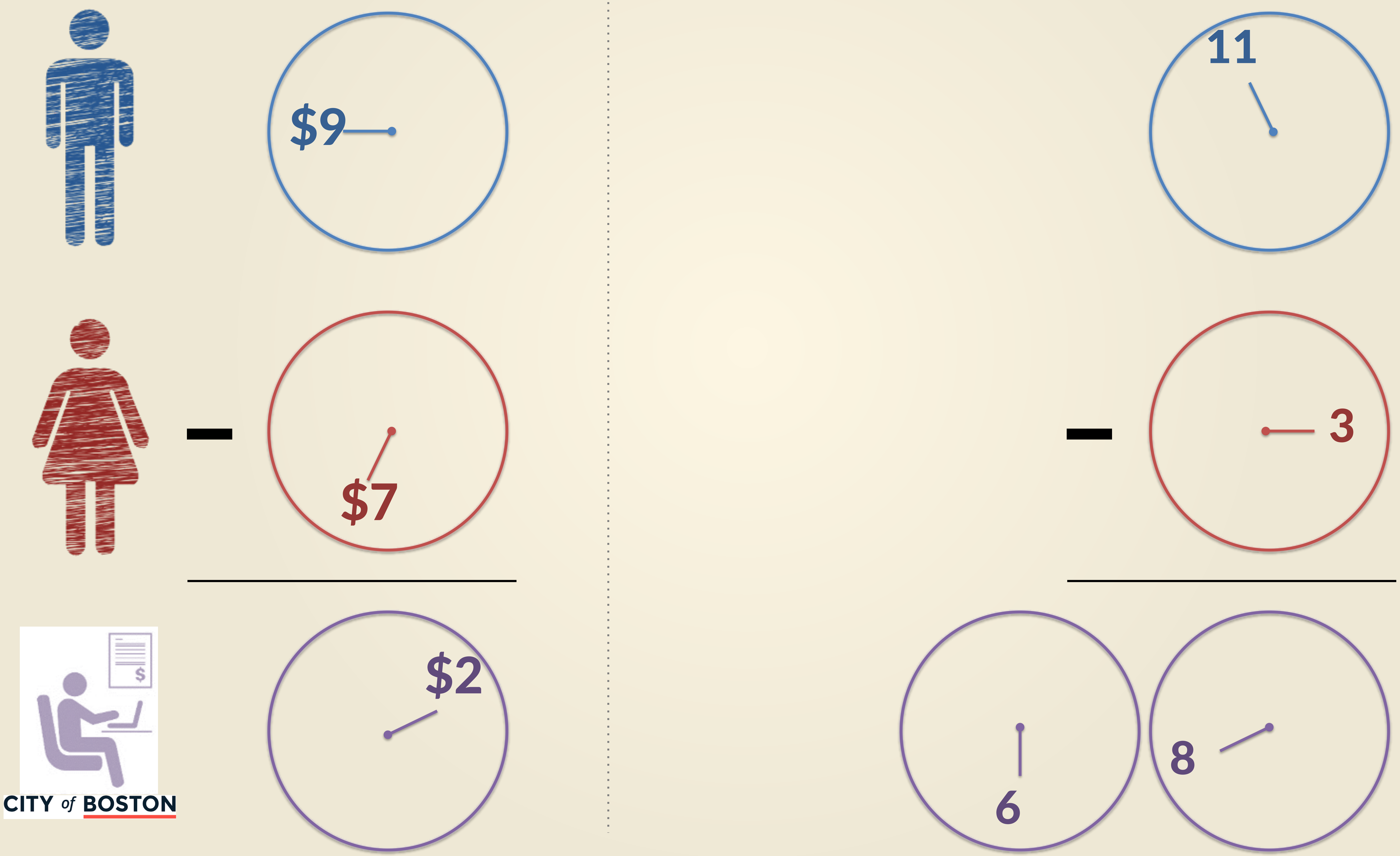


How it works



How it works

100% TALENT



13.2 Calculating Linear Functions

Another viewpoint: 3 steps to MPC

Computing servers

$$a = a_1 + a_2 \qquad b = b_1 + b_2$$

Split secrets from data contributors

$$y_1 = b_1 - a_1$$

$$y_2 = b_2 - a_2$$

Compute over secret shares

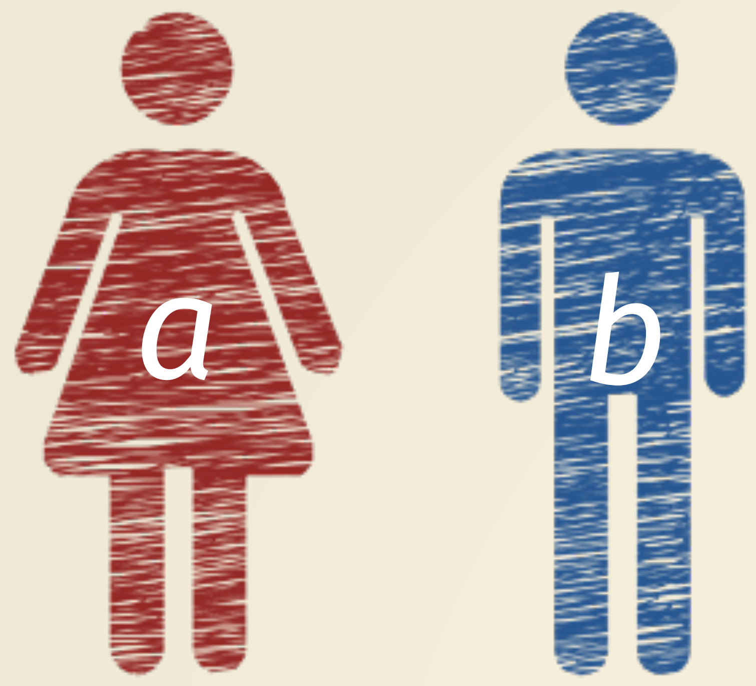
CITY of BOSTON

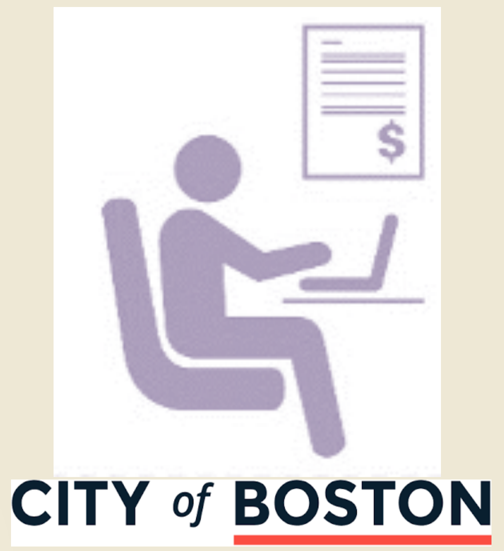
$$y = y_1 + y_2 = b - a$$

Reconstruct the final answer

Simpler notation

Generic server





$[a]$ $[b]$

$[y] = [a] + [b]$

open y

Secret share

Compute

Reconstruct

Adding secret + public value

Generic server

$[a]$ c

Secret share

$[y] = [a] + c$

Compute

open y

Reconstruct

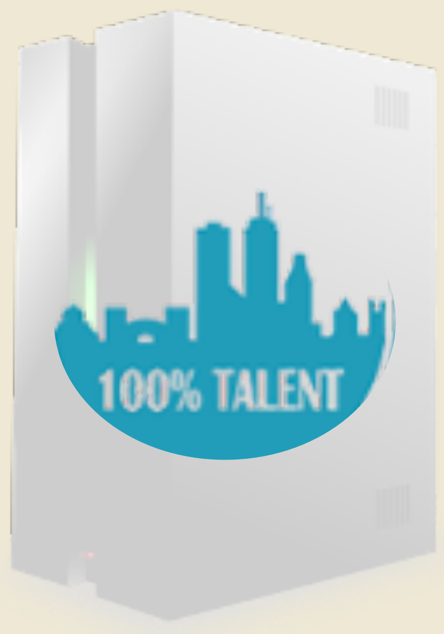
Adding secret + public value (in detail)

Computing servers

a_1 c

$y_1 = a_1 + c$

$$y = y_1 + y_2$$
$$= a + c$$



a_2 c

$y_2 = a_2$

Secret share | *Compute* | *Reconstruct*

Scalar multiplication

Generic server

$[a]$ c

Secret share

$[y]$ = c * $[a]$

Compute

open y

Reconstruct

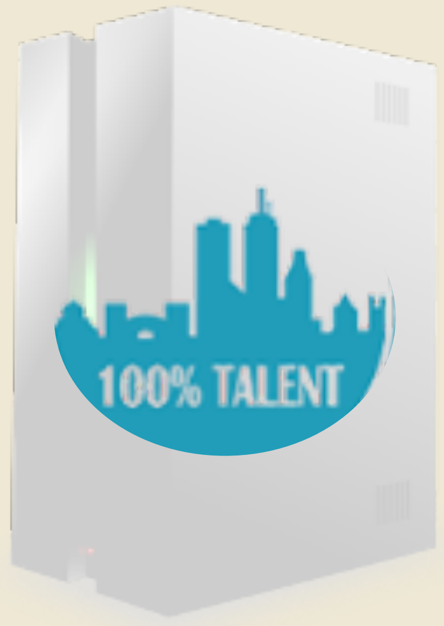
Scalar multiplication (in detail)

Computing servers

a_1 $\quad c$

$y_1 = c * a_1$

$$y = y_1 + y_2 = c * a$$



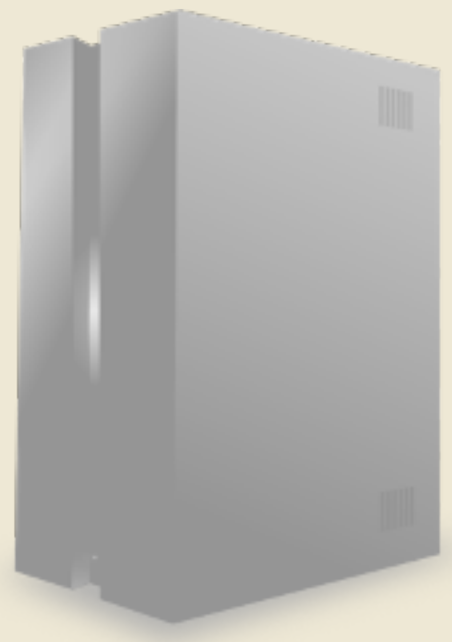
a_2 $\quad c$

$y_2 = c * a_2$

Secret share | *Compute* | *Reconstruct*

Extending to several inputs

Generic server



Secret share

$[a]$ $[b]$

$[c]$ $[d]$

e f

Compute

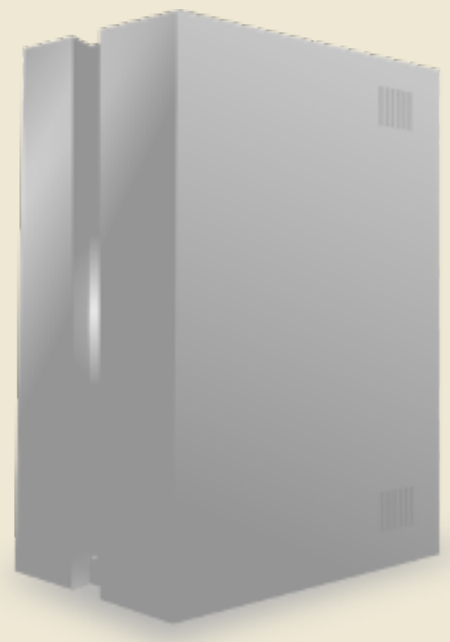
$[y]$ = e * $[b] + [d]$
- $[a] - [c] - f$

Reconstruct

open y from $[y]$

Upshot: can compute any linear function L!

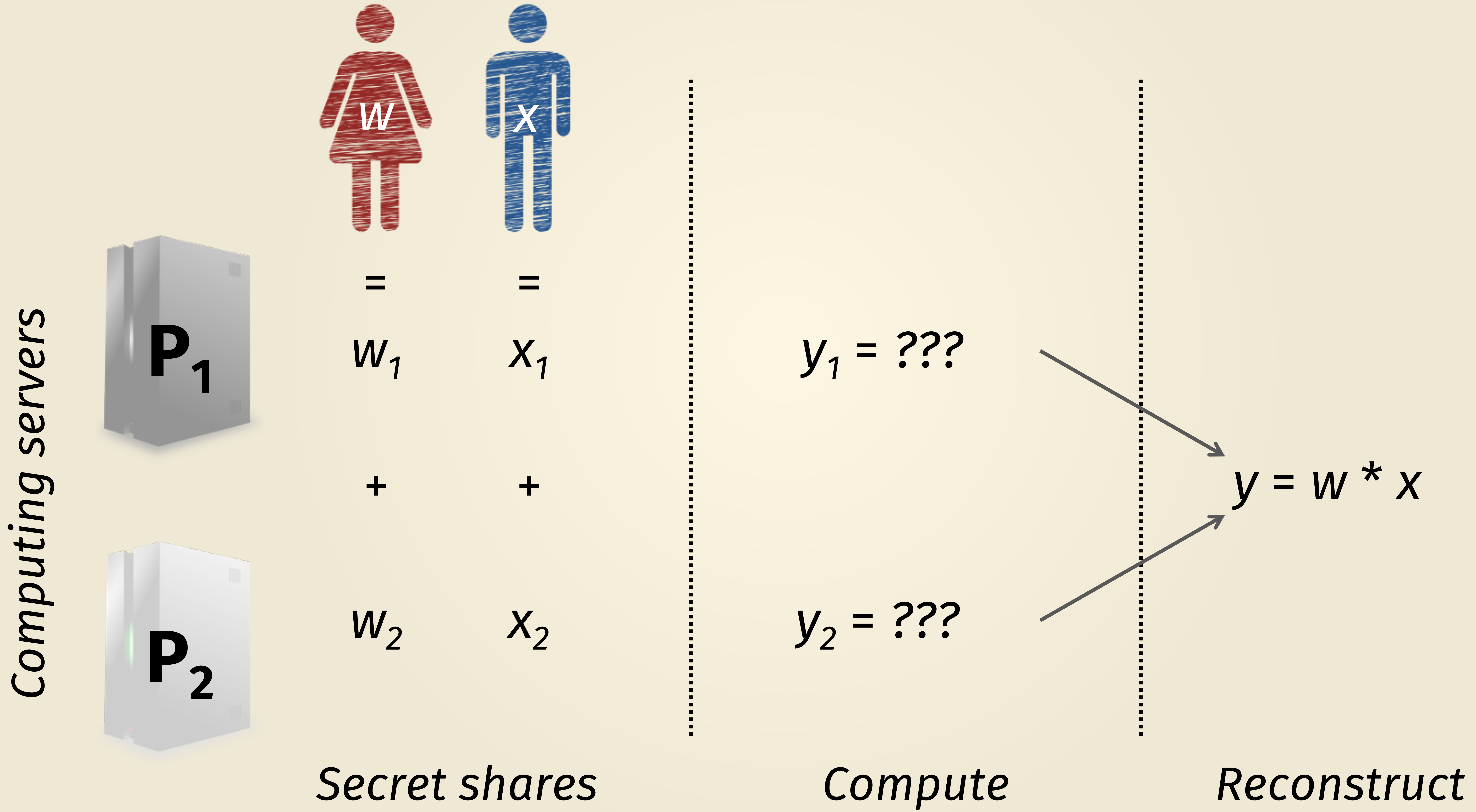
Generic server



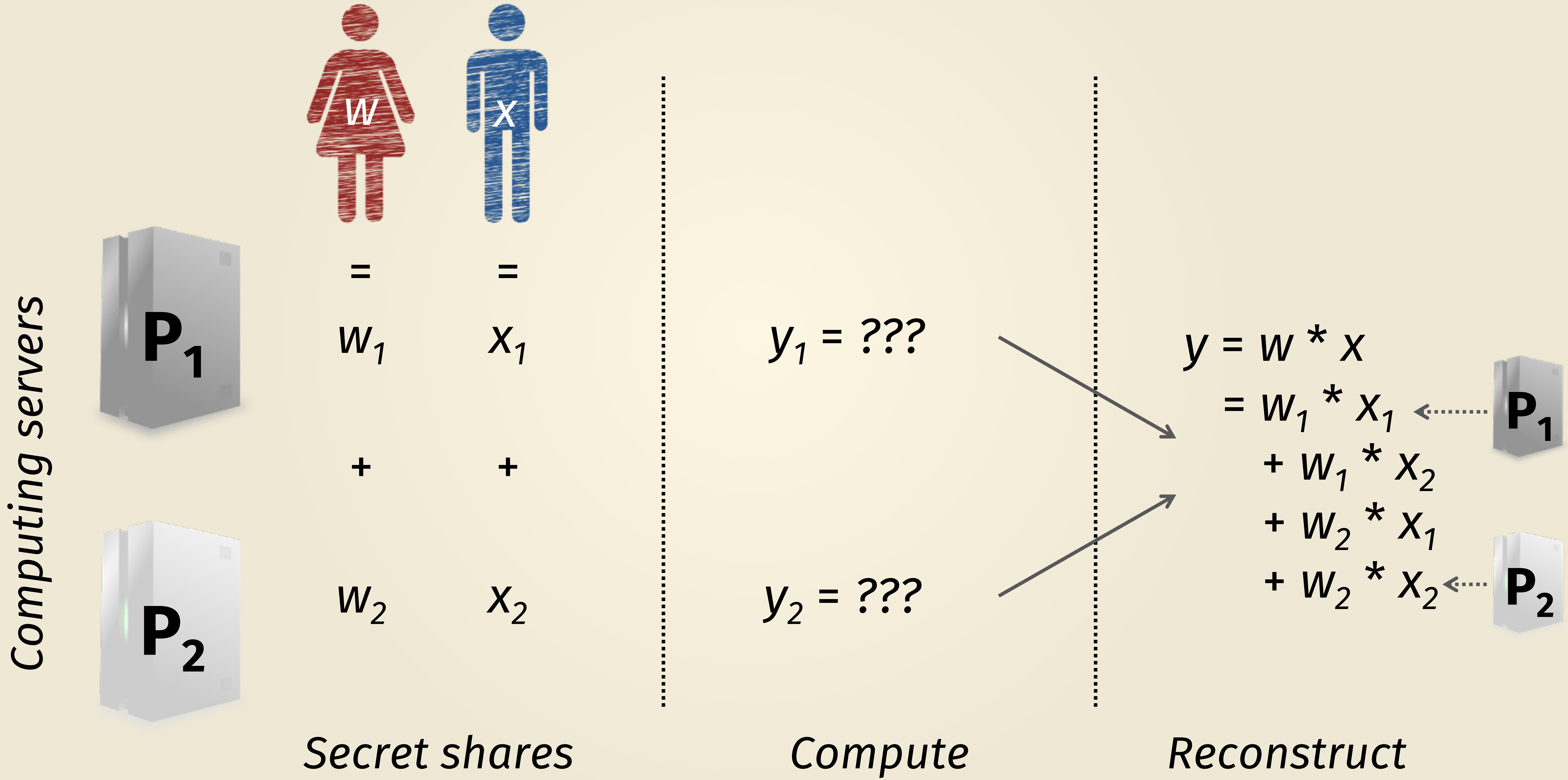
Secret share	*Compute*	*Reconstruct*
$[\boldsymbol{x}]$	$[y] = \mathrm{L}([\boldsymbol{x}])$	open y from $[y]$

13.3 Secure multiplication

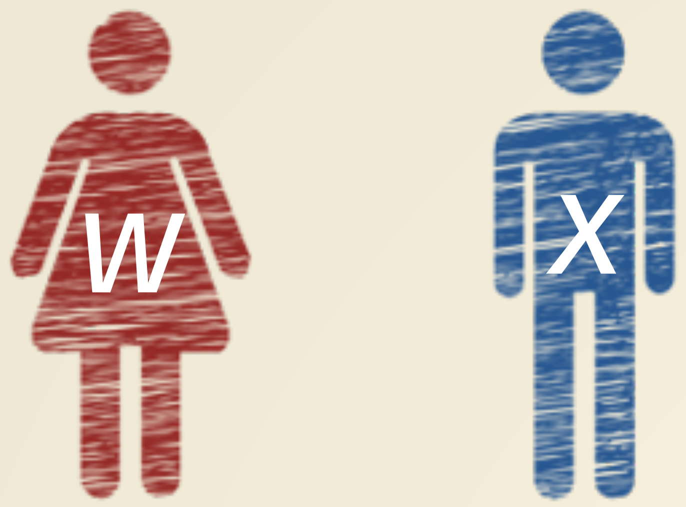
Can we multiply two secret variables?



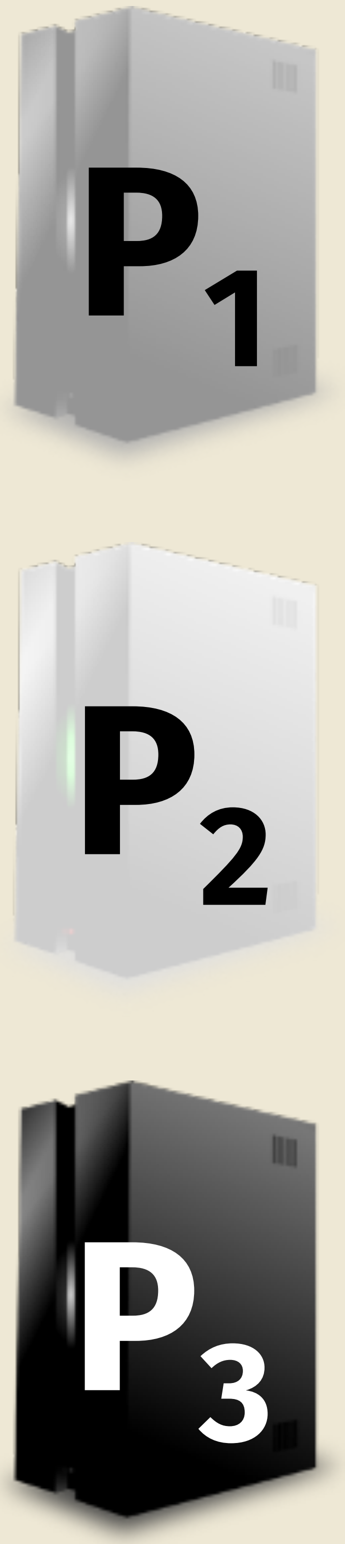
Can we multiply two secret variables?



Idea: add one more computing server



Computing servers



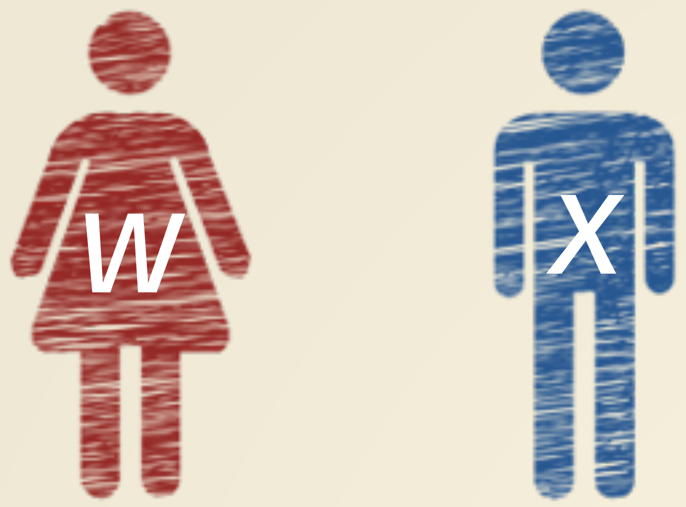
	Secret shares		Compute
	w	x	
	=	=	
P_1	w_1	x_1	y_1 = ???
	+	+	
P_2	w_2	x_2	y_2 = ???
	+	+	
P_3	w_3	x_3	y_3 = ???

Reconstruct

$$
\begin{aligned}
y &= w * x \\
&= w_1 x_1 + w_1 x_2 + w_1 x_3 \\
&\quad + w_2 x_1 + w_2 x_2 + w_2 x_3 \\
&\quad + w_3 x_1 + w_3 x_2 + w_3 x_3
\end{aligned}
$$

Secret shares *Compute* *Reconstruct*

Idea: give each computing server two shares



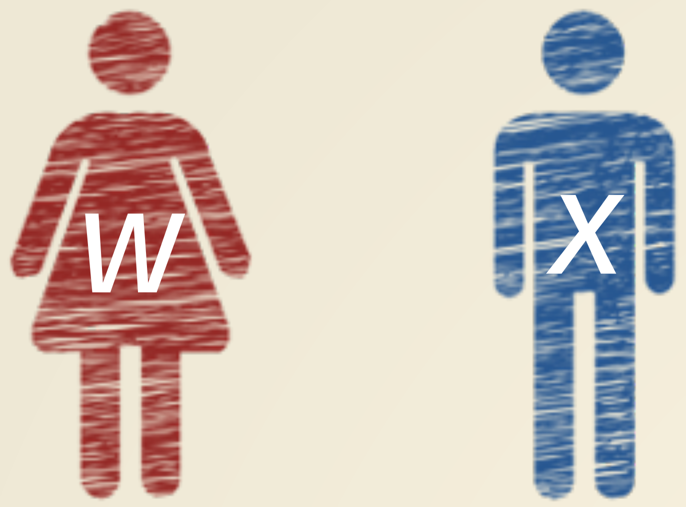
Computing servers

	Secret shares	Compute
P_1	w_1, w_2, x_1, x_2	y_1 = ???
P_2	w_2, w_3, x_2, x_3	y_2 = ???
P_3	w_3, w_1, x_3, x_1	y_3 = ???

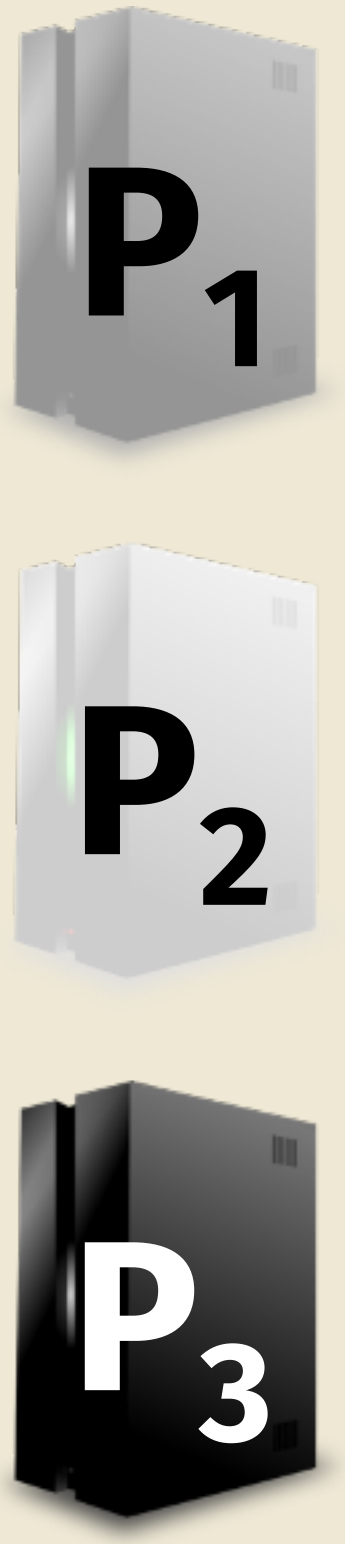
Reconstruct

$$
\begin{aligned}
y &= w * x \\
&= w_1 x_1 + w_1 x_2 + w_1 x_3 \\
&\quad + w_2 x_1 + w_2 x_2 + w_2 x_3 \\
&\quad + w_3 x_1 + w_3 x_2 + w_3 x_3
\end{aligned}
$$

Idea: give each computing server two shares



Computing servers



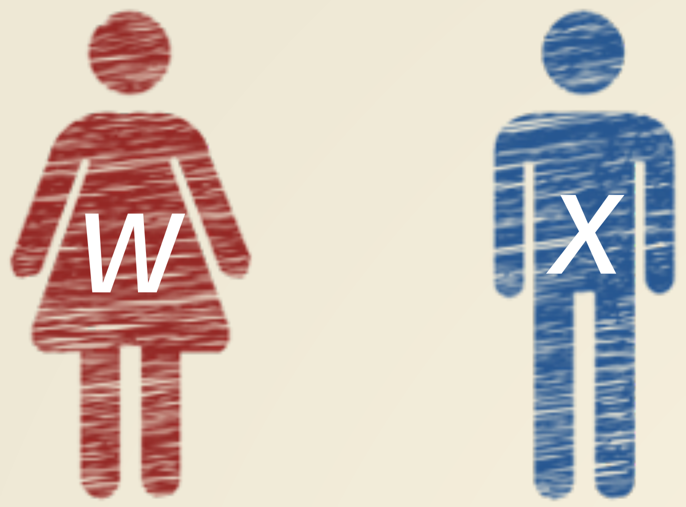
	Secret shares	Compute
P_1	w_1, w_2, x_1, x_2	$y_1 = w_1 x_2 + w_2 x_1 + w_1 x_1$
P_2	w_2, w_3, x_2, x_3	$y_2 = w_2 x_3 + w_3 x_2 + w_2 x_2$
P_3	w_3, w_1, x_3, x_1	$y_3 = w_3 x_1 + w_1 x_3 + w_3 x_3$

Reconstruct

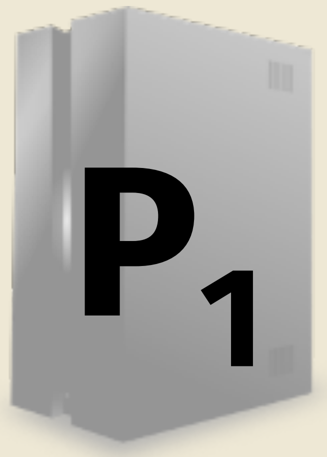
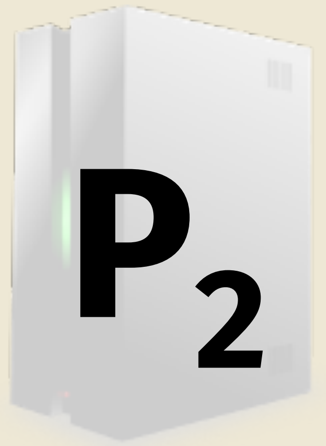
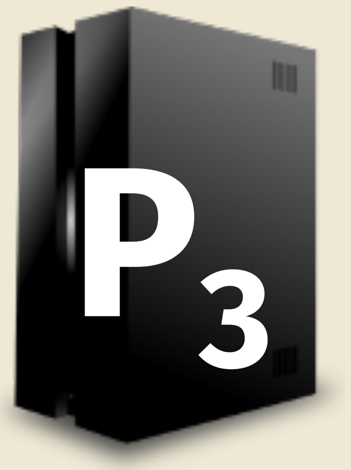
$$\begin{aligned} y &= w * x \\ &= w_1 x_1 + w_1 x_2 + w_1 x_3 \\ &\quad + w_2 x_1 + w_2 x_2 + w_2 x_3 \\ &\quad + w_3 x_1 + w_3 x_2 + w_3 x_3 \end{aligned}$$

Secret shares *Compute* *Reconstruct*

Idea: give each computing server two shares



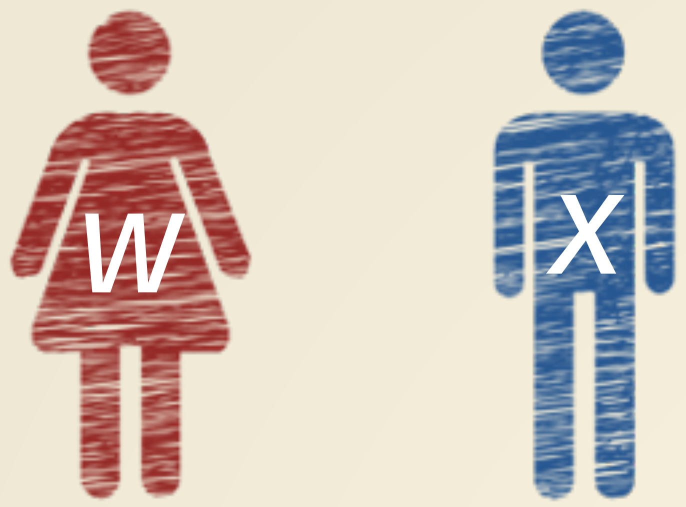
Computing servers

	Secret shares	Compute
	w_1, w_2, x_1, x_2	$y_1 = w_1 x_2 + w_2 x_1 + w_1 x_1$
	w_2, w_3, x_2, x_3	$y_2 = w_2 x_3 + w_3 x_2 + w_2 x_2$
	w_3, w_1, x_3, x_1	$y_3 = w_3 x_1 + w_1 x_3 + w_3 x_3$

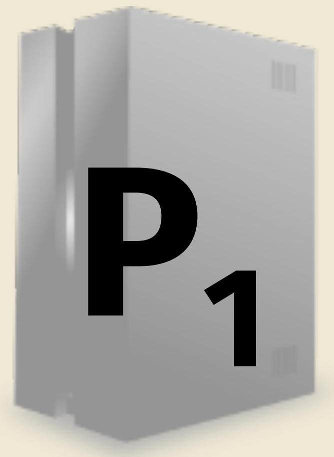
Reconstruct

$$y = w * x = y_1 + y_2 + y_3$$

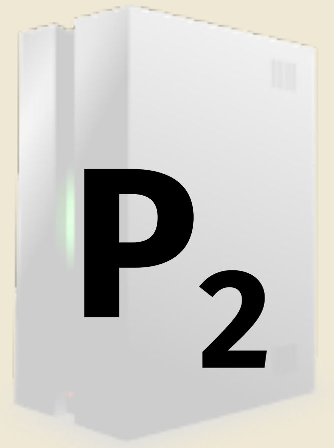
Idea: give each computing server two shares



Computing servers



w_1, w_2, x_1, x_2



w_2, w_3, x_2, x_3

w_3, w_1, x_3, x_1

Secret shares

$$y_1 = w_1 x_2 + w_2 x_1 + w_1 x_1$$

$$\downarrow$$

$$y_2 = w_2 x_3 + w_3 x_2 + w_2 x_2$$

$$\downarrow$$

$$y_3 = w_3 x_1 + w_1 x_3 + w_3 x_3$$

Compute

$$y = w * x = y_1 + y_2 + y_3$$

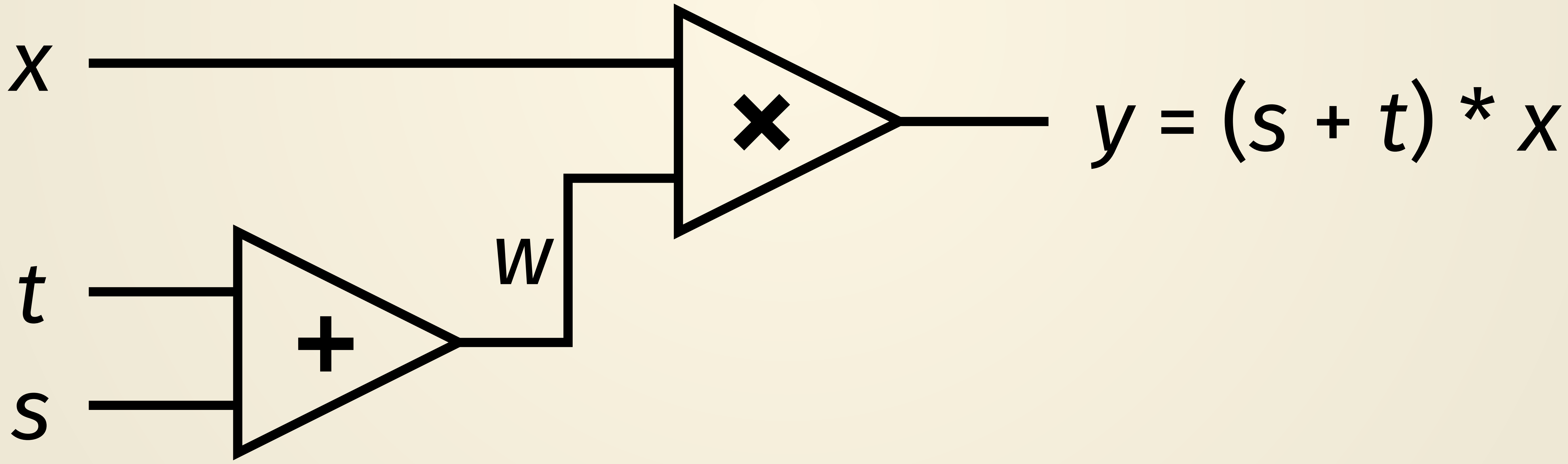
Reconstruct

Analysis of multiplication

- This technique works to multiply two secrets, without learning them!
- Invariant: each party maintains 2 of the 3 additive shares of each secret
- Correctness when adding secrets: same as before
- Correctness when multiplying secrets: each party computes 3 terms of the product y, as shown by the distributive property
- Security: any single party has no idea what the secret is since the final share could be anything… but note that the threshold $T = 1$ (not 2!)
- Efficiency: parties can do addition on their own, must talk to multiply

Secure computation of everything

- So far we have seen secure computation of +, -, and ×
- + and × are Turing-complete, so we can securely compute *any function*!
  - (This may not be the *fastest* way to compute *f* securely, however...)



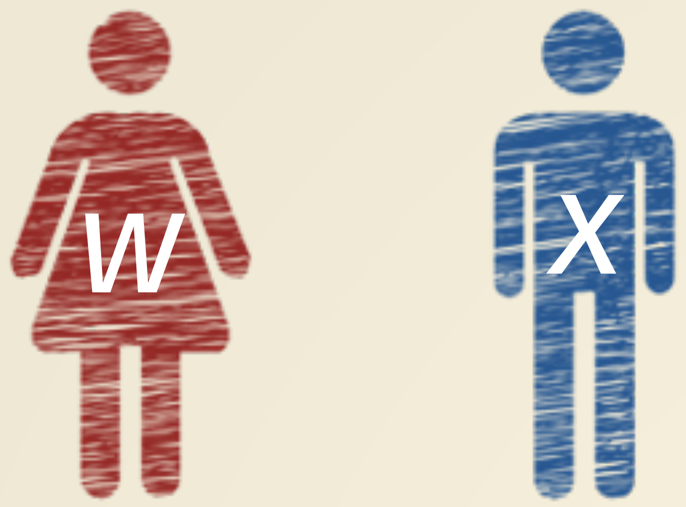
- For instance: given the circuit above and [*s*], [*t*], [*x*], the three computing parties can work together to calculate [*w*] and then [*y*], and only open *y*

13.4 Security against Byzantine compute parties

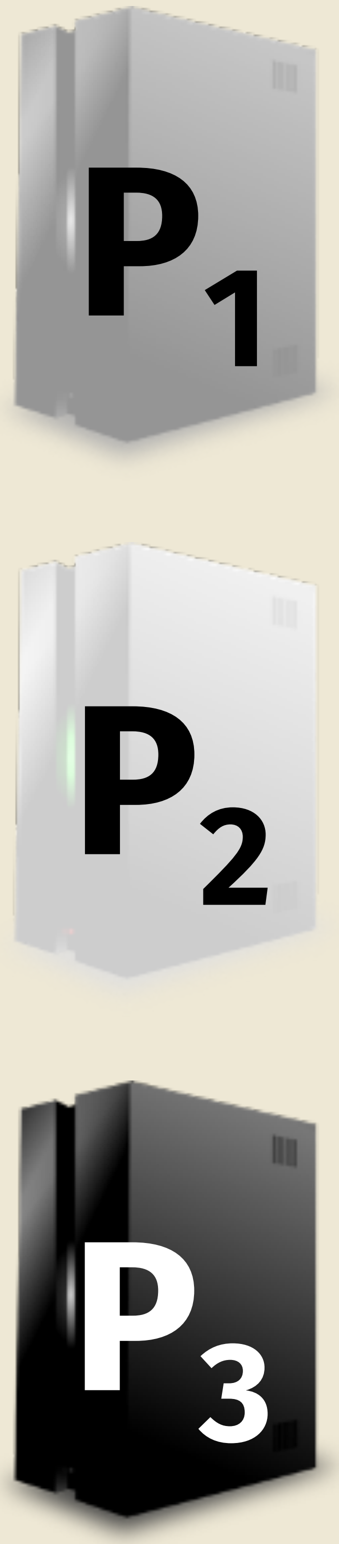
Reminder: our security objectives

- Our concern is that up to t of the n parties are adversarial
- We will consider 3 kinds of security guarantees to enforce
  - Semi-honest security: withstands an adversary who follows the protocol but is trying to learn data (i.e., break confidentiality)
  - Malicious security: withstands an adversary who also might deviate from the protocol to learn data or alter the results of the computation (break integrity)
  - Robustness: withstands an adversary who also might quit participation (break availability), and will reach agreement on the result of the computation anyway
    - (This is similar to “agreement” in the setting of asynchronous protocols)

The current protocol is only semi-honest!



Computing servers



w_1, w_2, x_1, x_2

w_2, w_3, x_2, x_3

w_3, w_1, x_3, x_1

Secret shares

$$y_1 = w_1 x_2 + w_2 x_1 + w_1 x_1$$

$$\downarrow$$

$$y_2 = w_2 x_3 + w_3 x_2 + w_2 x_2$$

$$\downarrow$$

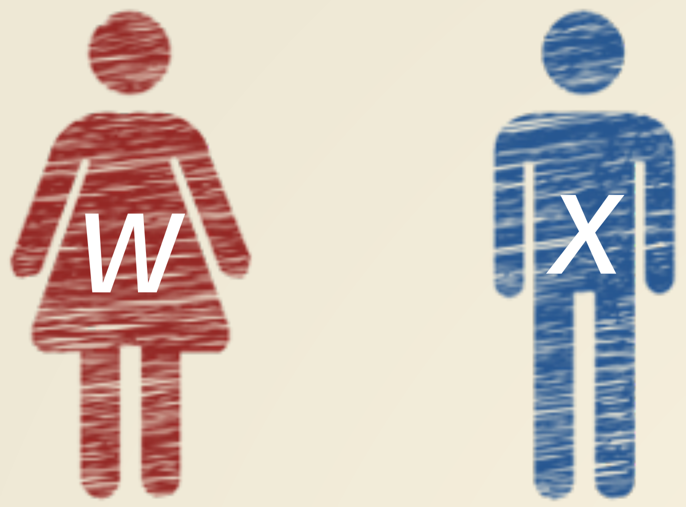
$$y_3 = w_3 x_1 + w_1 x_3 + w_3 x_3$$

Compute

$$y = w * x = y_1 + y_2 + y_3$$

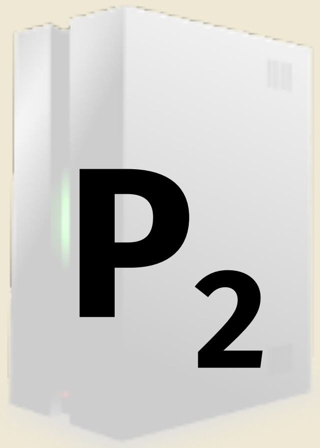
Reconstruct

Undetectable attack by a malicious party

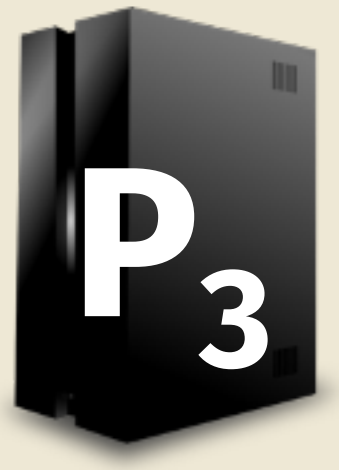


Computing servers

w_1, w_2, x_1, x_2



w_2, w_3, x_2, x_3



w_3, w_1, x_3, x_1

$y_1' = y_1 + \Delta$

$\downarrow$

$y_2 = w_2 x_3 + w_3 x_2 + w_2 x_2$

$\downarrow$

$y_3 = w_3 x_1 + w_1 x_3 + w_3 x_3$

$y' = y + \Delta$

$= y_1 + \Delta + y_2 + y_3$

Secret shares | *Compute* | *Reconstruct*

Idea: add yet one more computing server

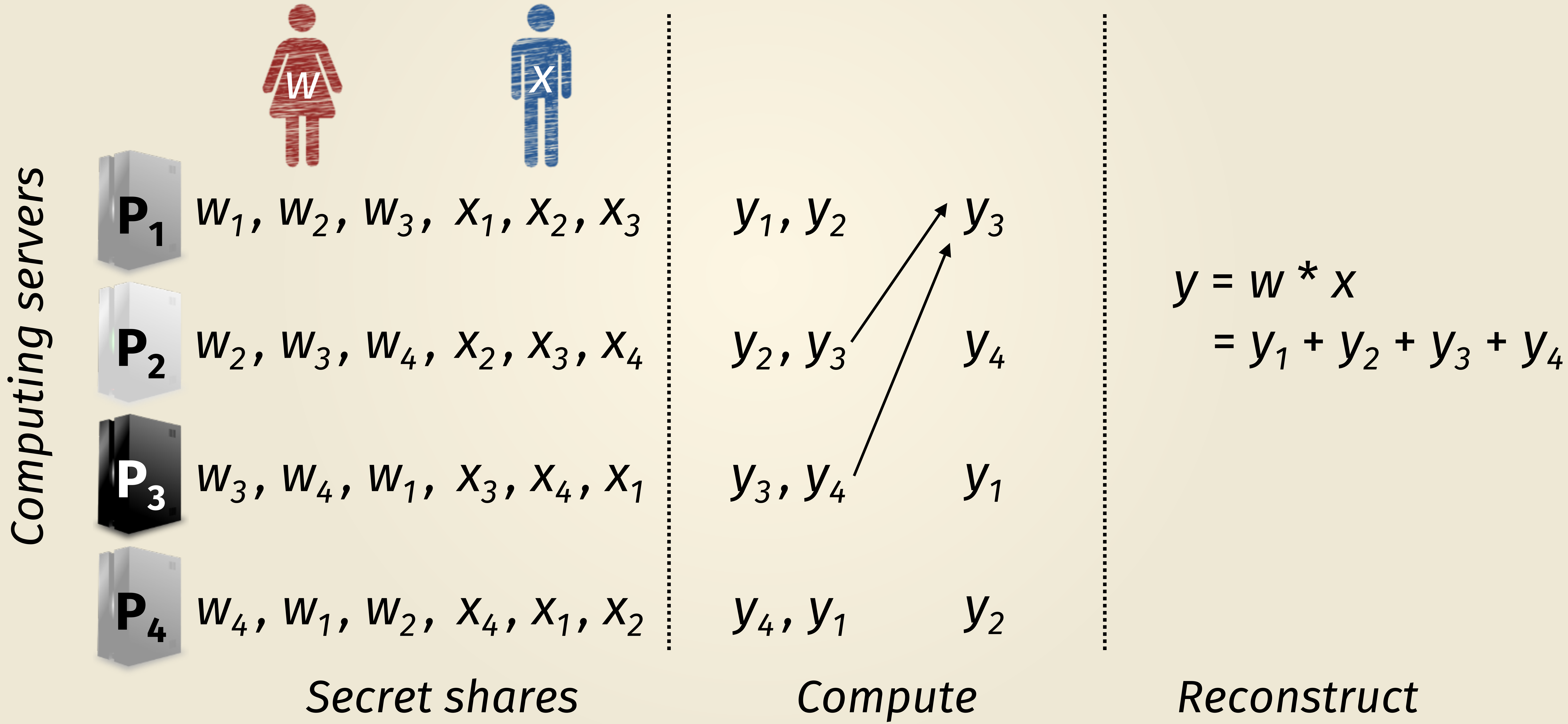
Computing servers

	w	x	
	=	=	
P_1	w_1	x_1	$y_1 = ???$
	+	+	
P_2	w_2	x_2	$y_2 = ???$
	+	+	
P_3	w_3	x_3	$y_3 = ???$
	+	+	
P_4	w_4	x_4	$y_4 = ???$

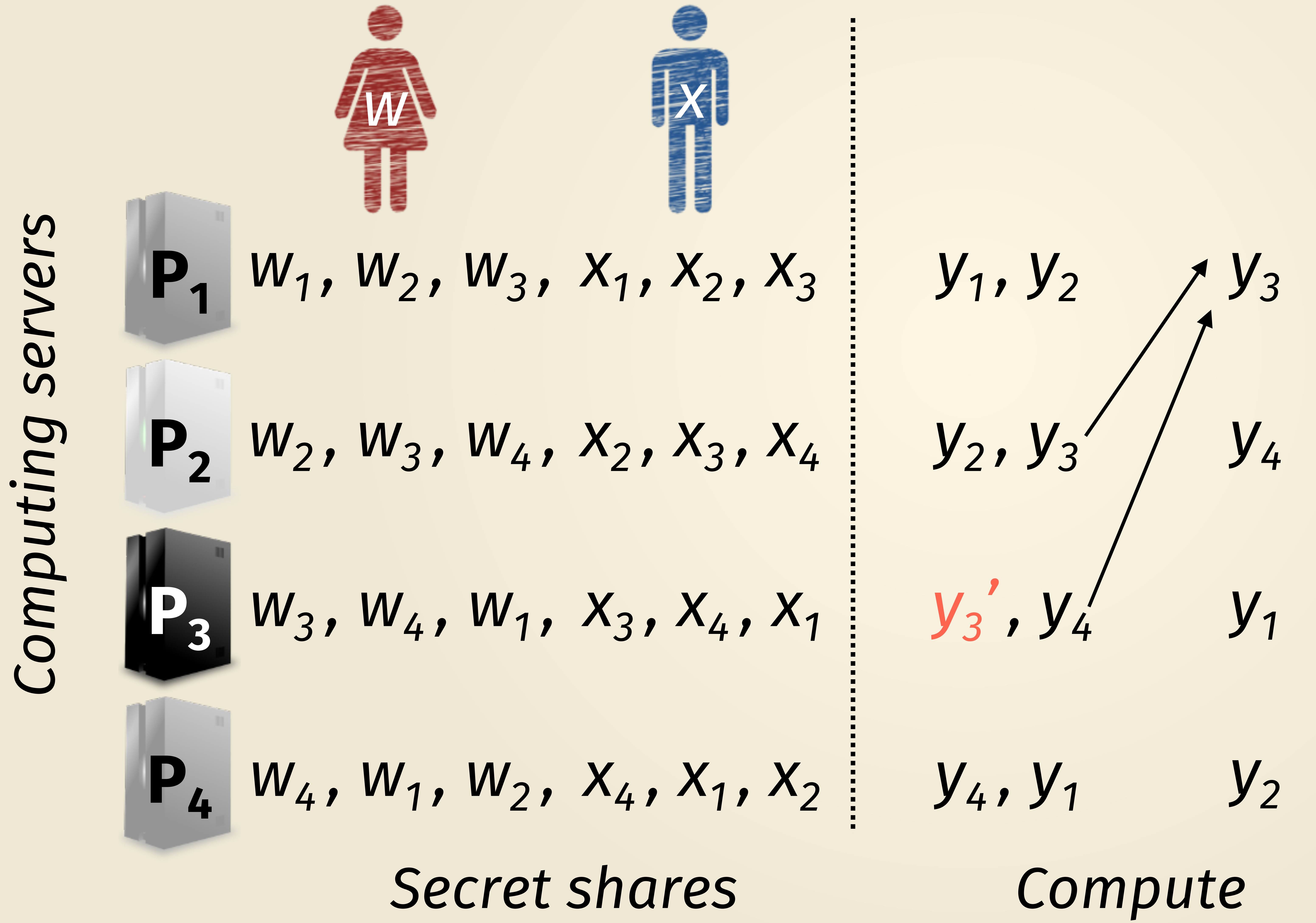
$$y = w * x$$
$$= y_1 + y_2 + y_3 + y_4$$

Secret shares | Compute | Reconstruct

Compute and send as before, now with redundancy!



Redundancy → detect errors



$$y = w * x$$
$$= y_1 + y_2 + y_3 + y_4$$

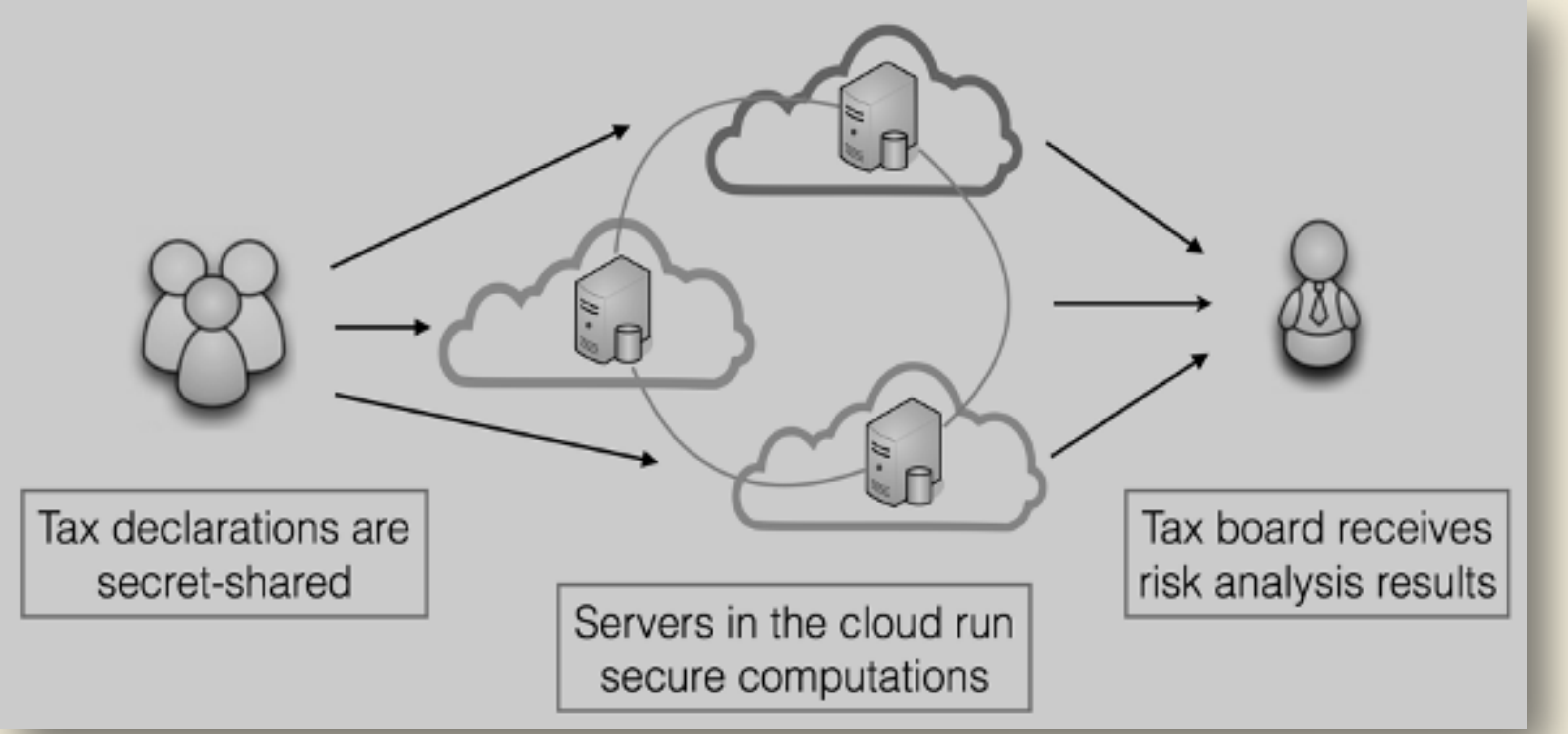
Reconstruct

Security analysis

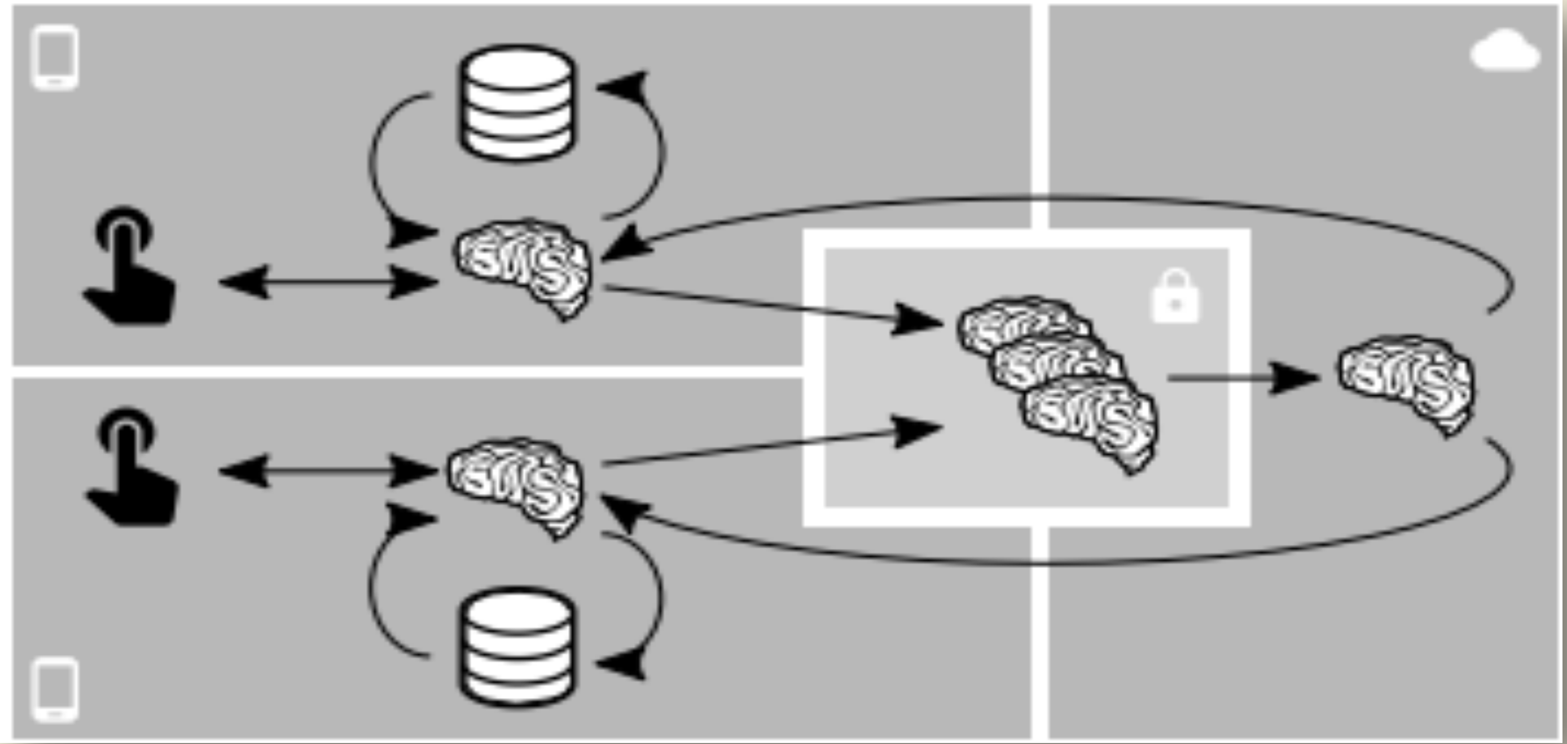
- Bad news: this protocol has a worse threshold: $T = 1$ of $N = 4$ parties
- Good news: security holds even against *malicious adversaries* who don't obey the rules of the protocol
  - Furthermore, we've narrowed down the adversary to one of two parties
  - Achieve *robustness* by switching to a semi-honest secure protocol with $N = 2$
- Upper bounds on what's possible, with more sophisticated crypto:
  - Can achieve semi-honest or malicious security against $T = N - 1$ parties
  - Can achieve robustness against $T < N / 2$ parties (intuition: just as with Byzantine agreement, need an honest majority to vote on the correct answer)

Some deployments of MPC in practice

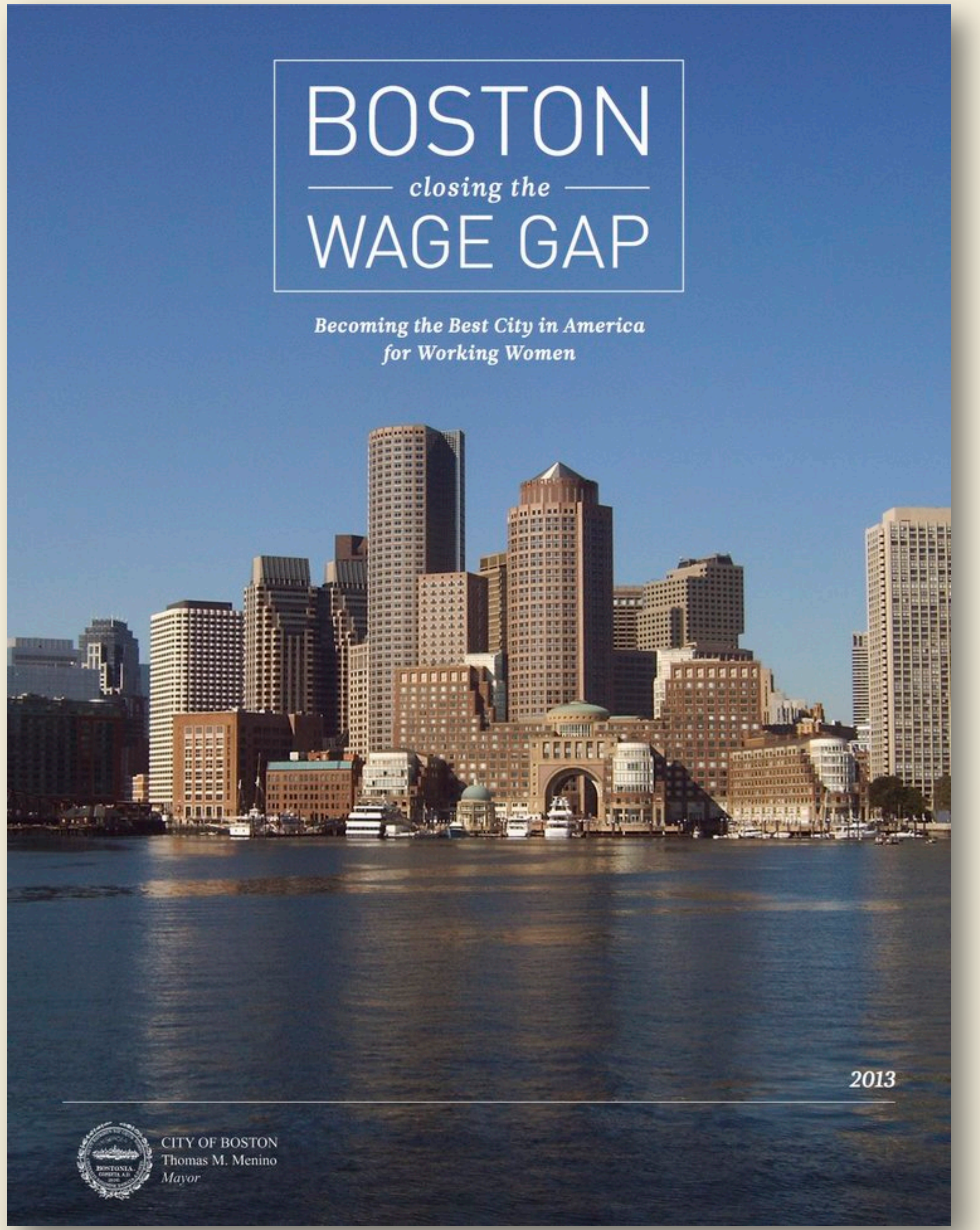
Cybernetica: VAT tax audits



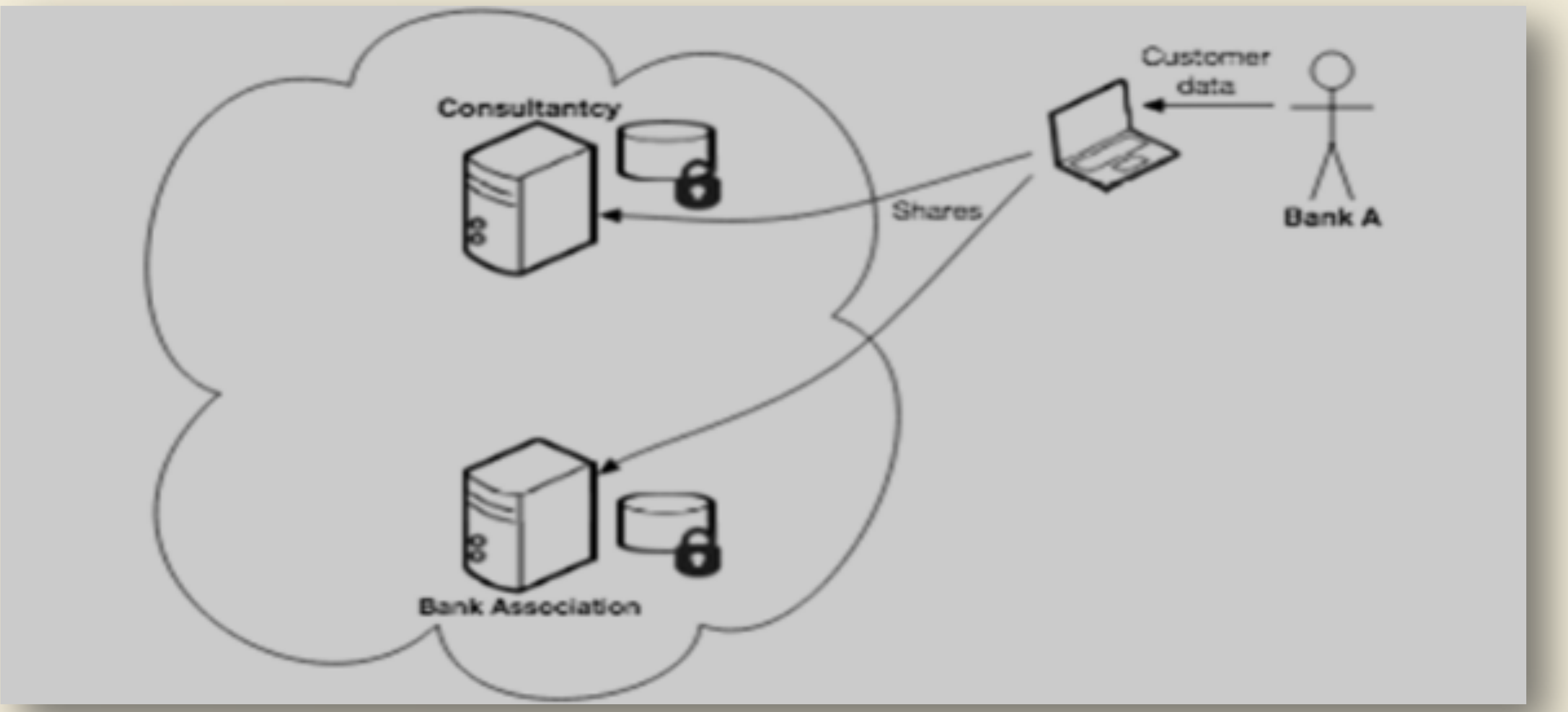
Google: Federated machine learning



BU: Pay equity in Boston



Partisia: Rate credit of farmers



Unbound: Protect cryptographic keys