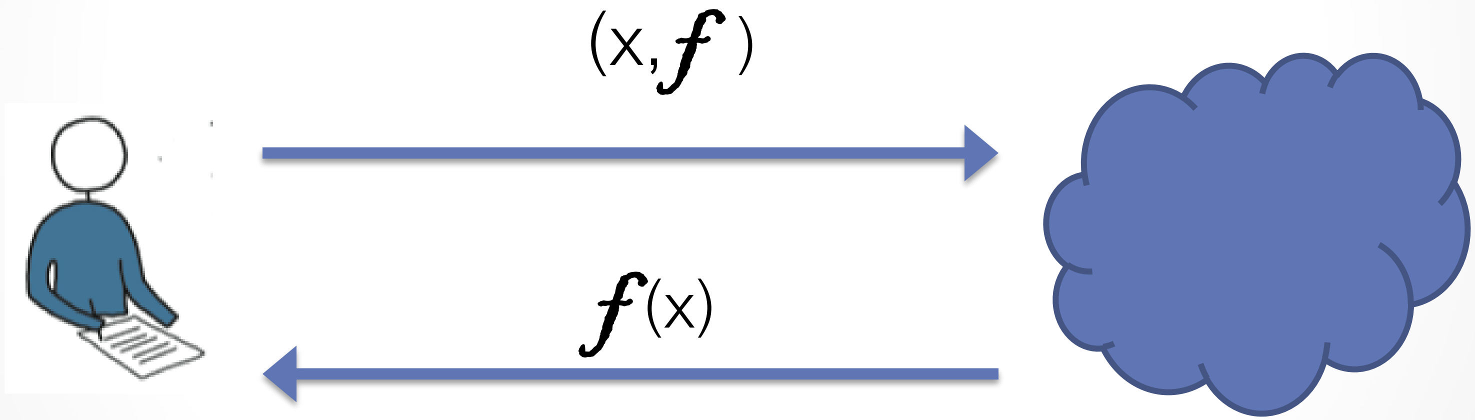
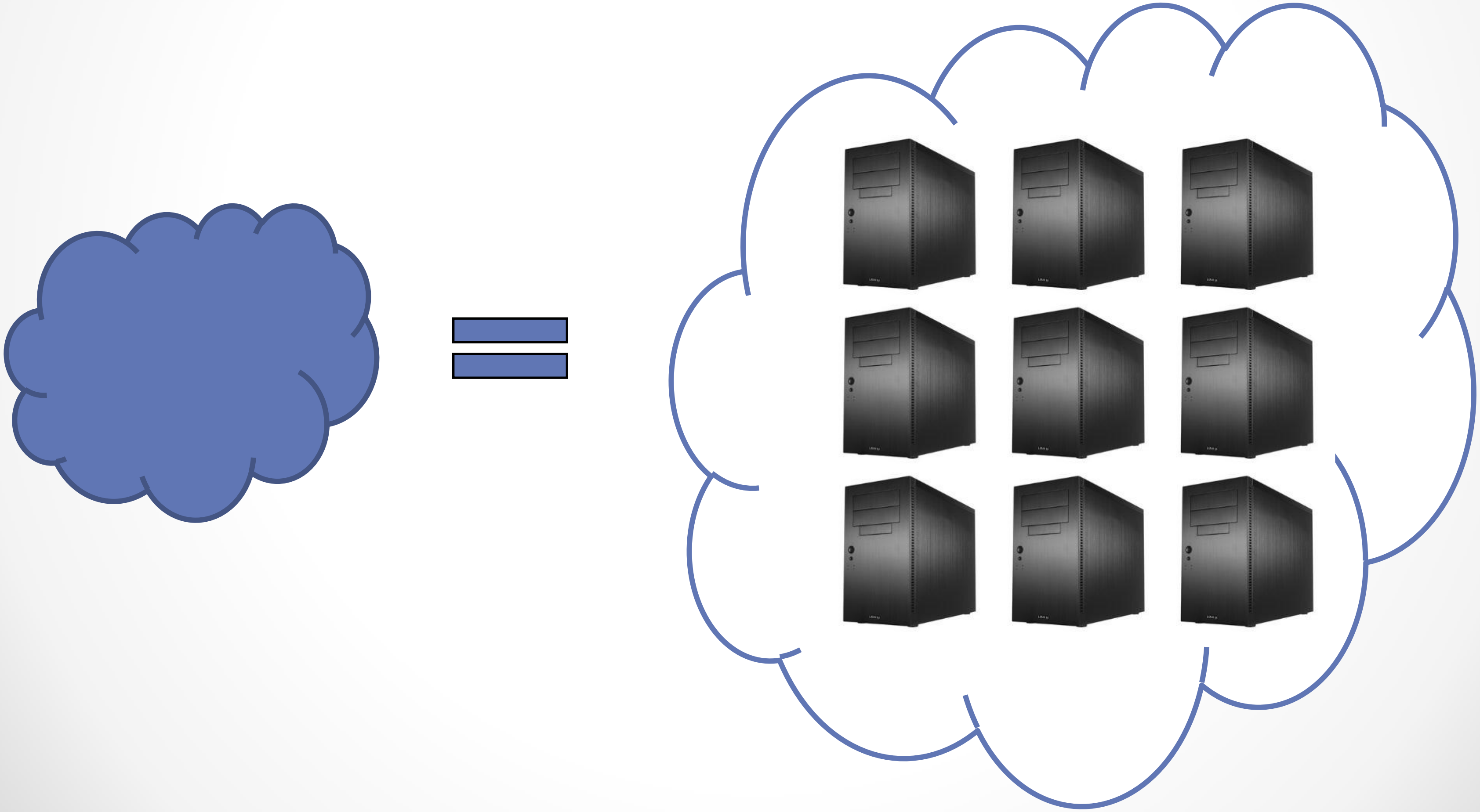
Secure Outsourced Computation in a Multi-Tenant Cloud

Seny Kamara - Microsoft Research
Mariana Raykova - Columbia

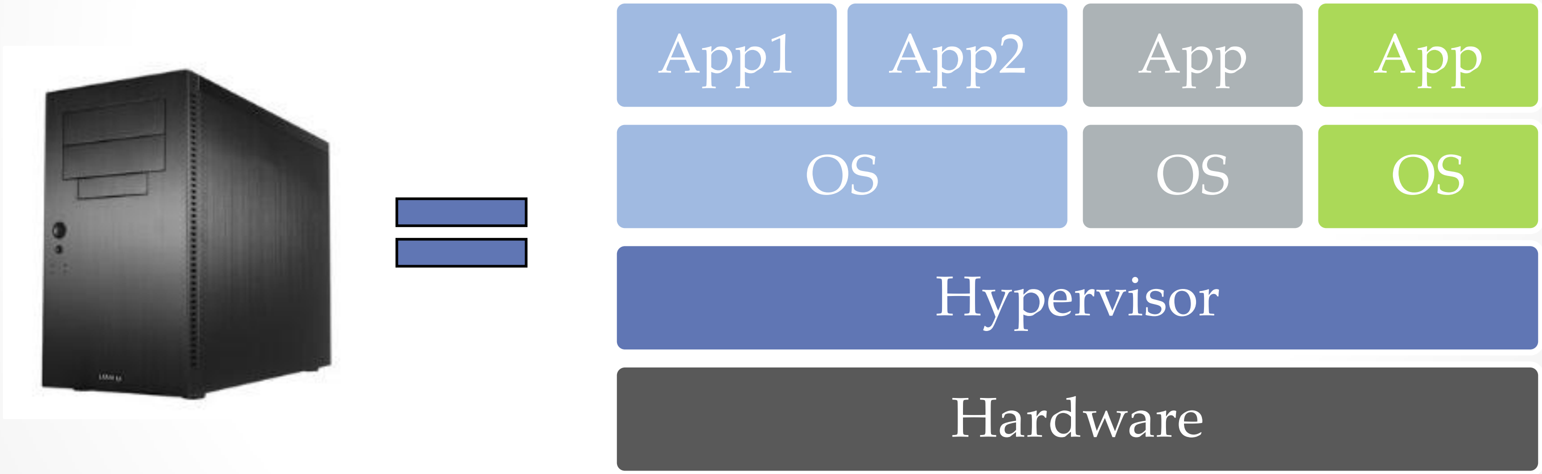
Outsourced Computation



The Cloud

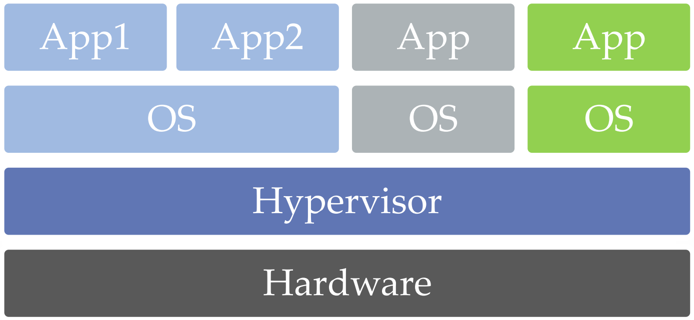


Virtualized Servers



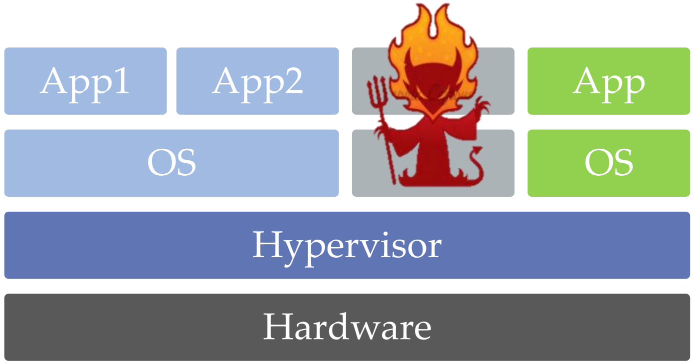
Multi-Tenancy

- Virtualization enables multi-tenancy
  - VMs from different clients run on the same server
- Multi-tenancy allows cloud operator to
  - Optimize resources usage
- This all leads to $ saved for clients



Multi-Tenancy

- Multi-tenancy is indispensible to cloud computing…
  - This is where part of the economic incentives come from
- …but it introduces security concerns
  - What if a co-located VM attacks my VM?
- Current solution is VM isolation
  - VMs cannot see each other's memory or state
  - Resources are appropriately shared



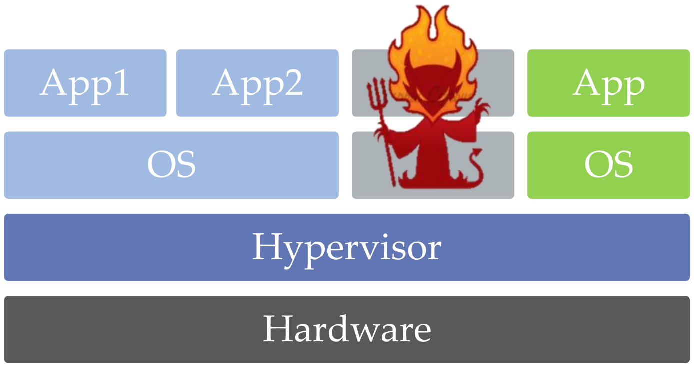
Breaking Isolation

- Exploiting the hypervisor
  - Some attacks known against VMware's ESX, XBOX's hypervisor, …
- Bypassing the hypervisor
  - [Ristenpart et al. 09] show that cross VM side-channels are possible

- Conclusion from [Ristenpart et al. 09]:

If security is a concern, use a single-tenant server.

How do we Protect vs. Multi-Tenancy?



VM Isolation

☹ vulnerabilities
☹ side-channels

Cryptography

☺ strong security

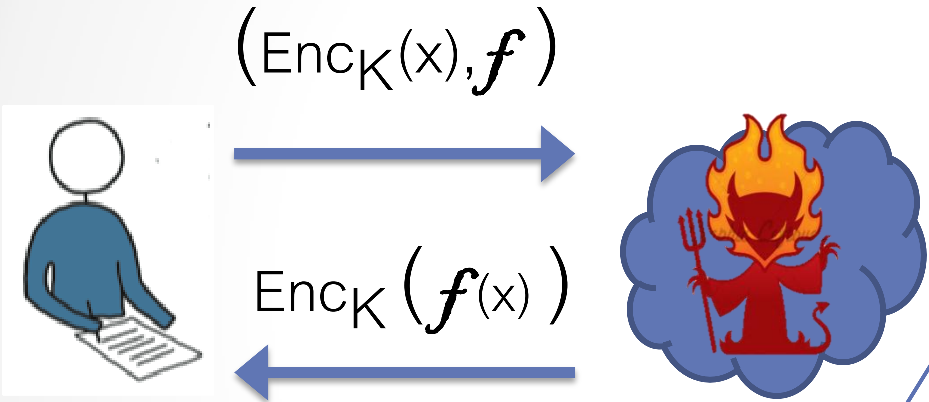
Single-Tenancy

☺ Perfectly secure

Outline

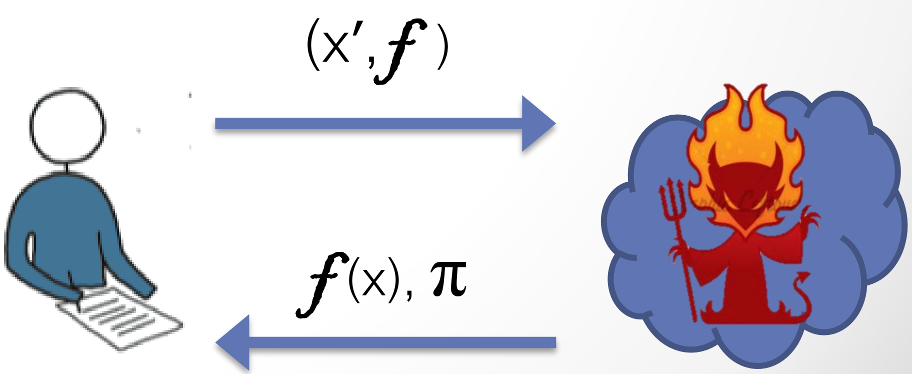
- Motivation
  - Secure outsourced computation in a multi-tenant cloud
- Delegation protocols
  - Security definition in ideal/real world paradigm
- General-purpose delegation protocol
  - Secret sharing & MPC
- Limitations of our approach

A Possible Approach



Fully-Homomorphic Encryption
[Gentry,…]

Verifiable computation
[Gentry-Gennaro-Parno, Chung-Kalai-Vadhan]

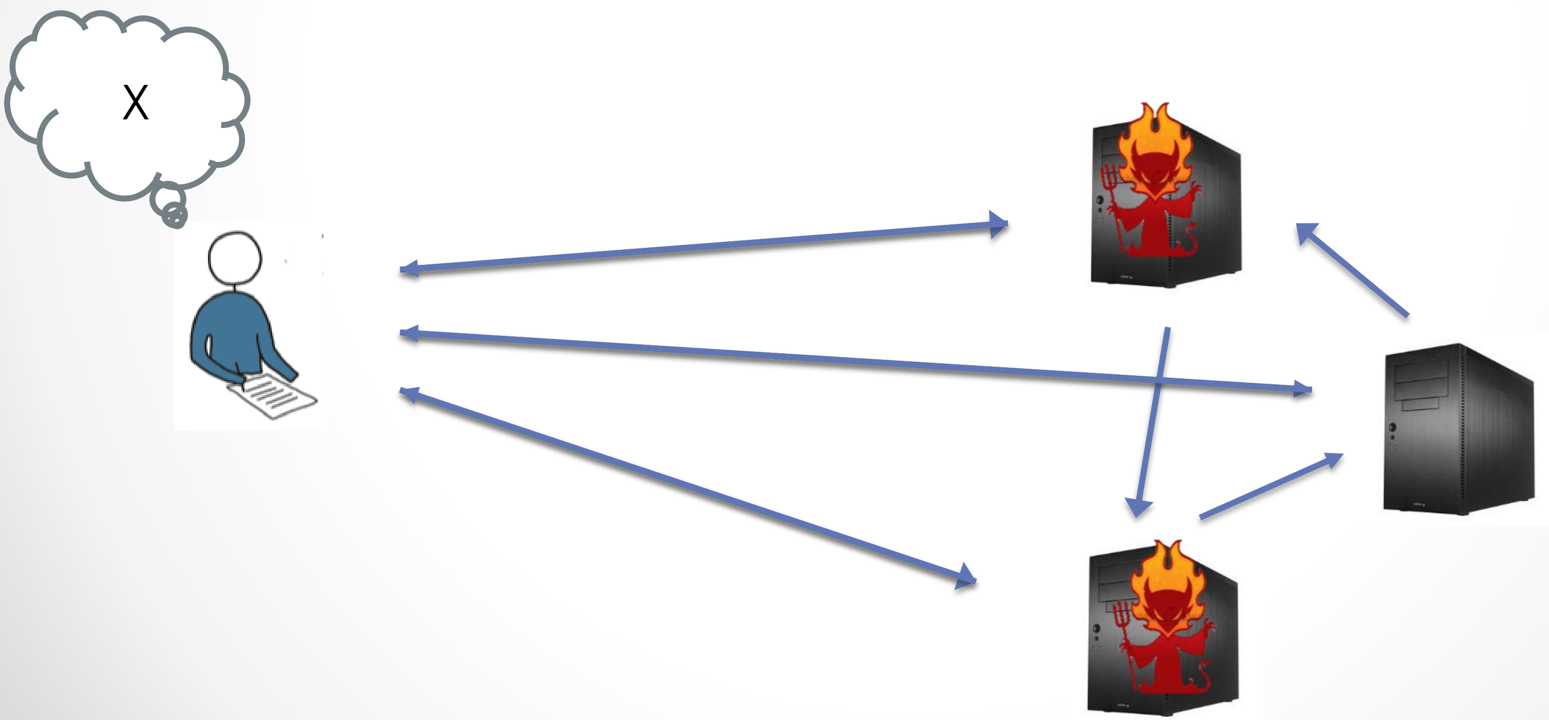


FHE + VC

- Efficiency
  - FHE is not practical
  - VC is based on FHE
- Overkill
  - Interaction is OK
  - *Cloud is not a single-server environment*

Delegation Protocol

- Protocol between
  - **C**: the client who provides an input
  - **VM_1,…,VM_w**: VM workers who have no input but return an output



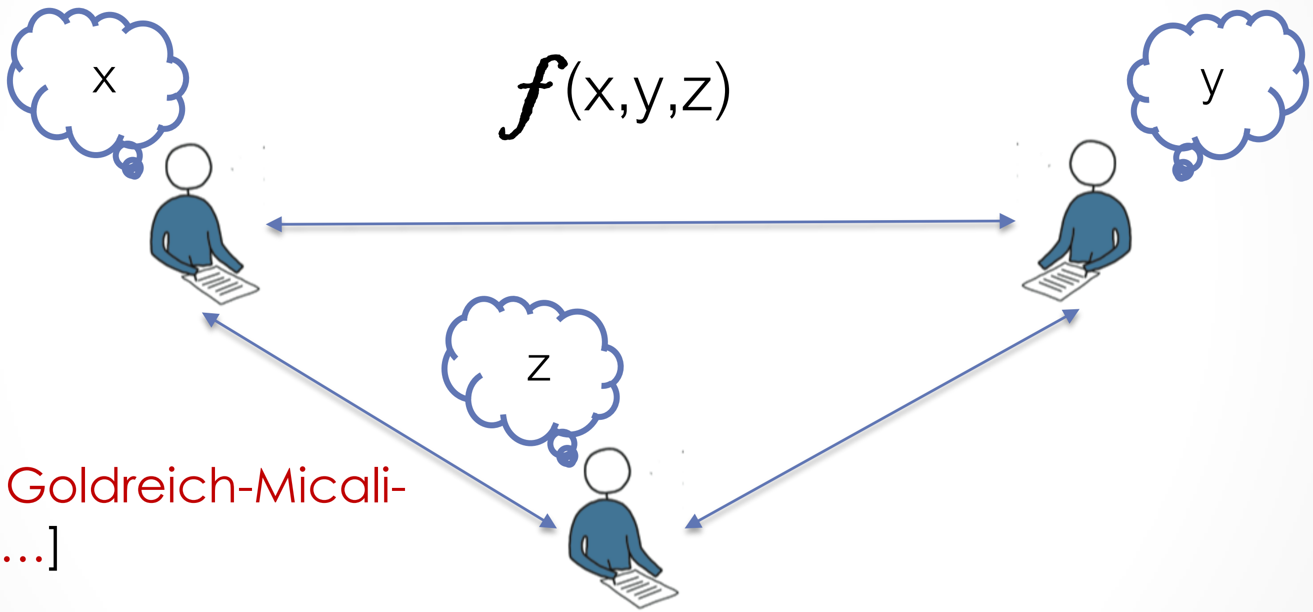
Underlying Assumption

- Cross VM attacks always work
  - **Semi-honest**: if $\mathcal{A}$ co-locates a VM then it recovers client VM's state
  - **Malicious**: if $\mathcal{A}$ co-locates a VM then it controls client VM
- Worst-case assumption
  - Makes our results stronger
  - Captures concerns of highly sensitive clients (e.g., governments)
- Not essential to our model
  - probability of successful cross VM attack can be taken into account

Security Definition

- Ideal/real world paradigm from MPC [...,Canetti01]
  - **Real execution**: **C** and VMs run the real protocol in presence of $\mathcal{A}$ that can co-locate adversarial VMs
  - **Ideal execution**: **C** sends input to trusted party who returns **f(x)** in presence of $\mathcal{A}$ that can co-locate adversarial VMs
  - **Security**: "every $\mathcal{A}$ in the real world can be emulated by an $\mathcal{A}'$ in the ideal world"
  - Note: If $\mathcal{A}$ is malicious then it is allowed to abort during the executions
- Guarantees:
  - *As long as* $\mathcal{A}$ *co-locates at most (w - 1) adversarial VMs*
  - **Privacy**: $\mathcal{A}$ learns no information about **C**'s input or output
  - **Correctness**: **C** receives correct output

Multi-Party Computation

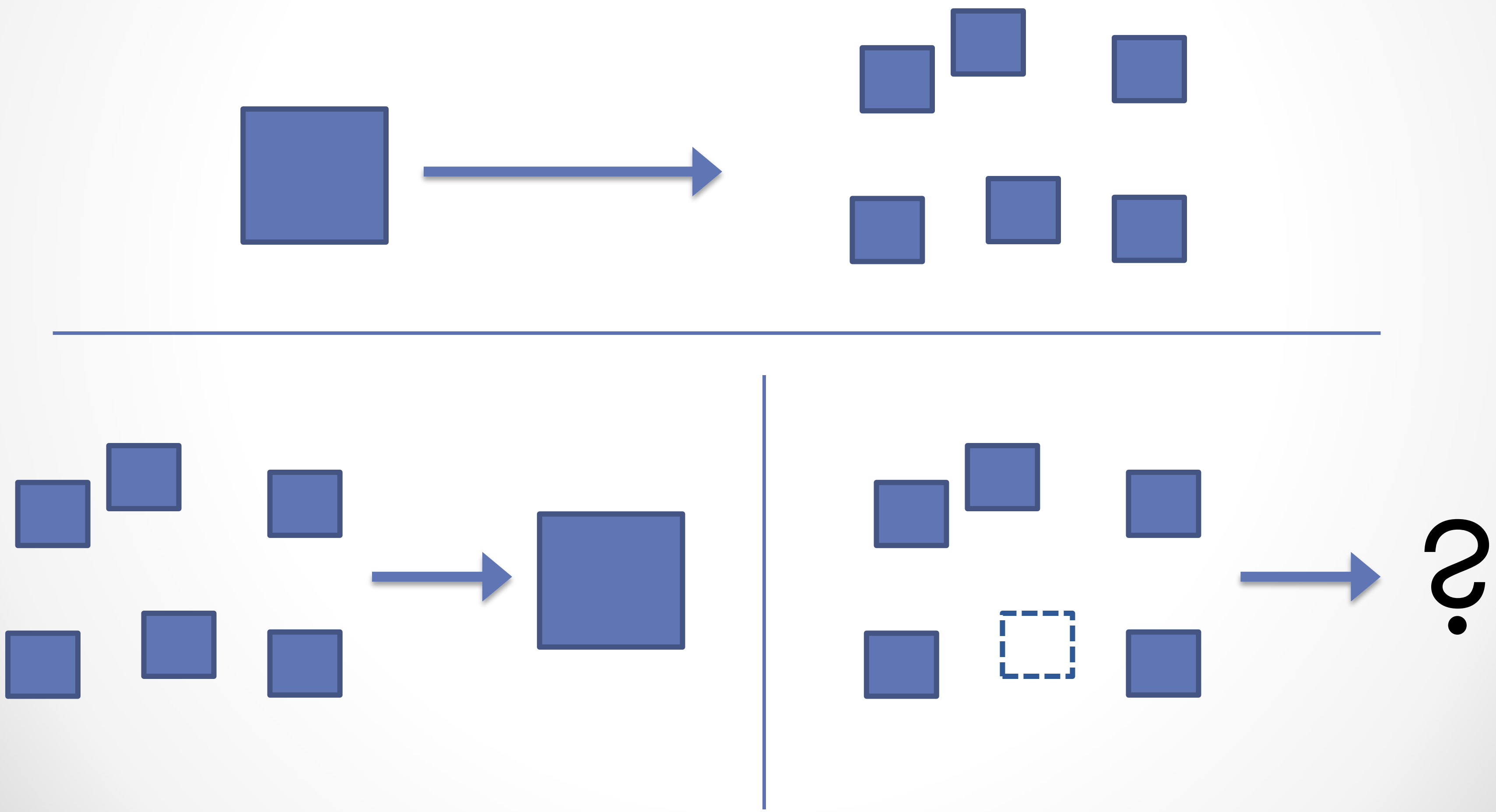


80's: [Yao, Goldreich-Micali-Wigderson,…]

Today: [Mohassel-Franklin, Lindell-Pinkas, Kolesnikov-Sadeghi-Schnedier,…]

Secret Sharing

[Shamir]



A General-Purpose Protocol

- The approach
  - Split input **x** into w shares (**$s_1, \ldots, s_n$**)
  - Store each share in a *separate* VM
  - Make the VMs evaluate **F** using MPC
  - **F**($s_1, \ldots, s_n$; $r_1 \oplus \ldots \oplus r_n$):
    - recovers the input **x** from the shares
    - Evaluates **y** = **f(x)**
    - Use $r_1 \oplus \ldots \oplus r_n$ to generate w shares of **y**
    - Output a share of **y** to each VM
  - VMs send back their shares to **C** who recovers **y**

Intuition

- Secret sharing
  - $\mathcal{A}$ must corrupt each worker
- MPC
  - Enables VMs to securely compute on shared input
  - *Without revealing information about shares to other workers*
    - Prevents $\mathcal{A}$ from learning about 2+ shares with a single corruption
- "Coin Tossing"
  - Coins will be uniform as long as at least one worker is uncorrupted
  - Guarantees sharing of output is secure
- Delegation is secure vs. malicious $\mathcal{A}$ if MPC is

Limitations of Delegation Protocols

- Efficiency
  - Overhead for recover & share
  - Overhead for MPC [+ ZKPs/C&C if $\mathcal{A}$ is malicious]
- Cost
  - Requires an extra (n – 1) VMs
- Useful
  - if cost of protocol < cost of single-tenant server
- Ongoing work
  - *Efficient* delegation protocols for specific functionalities (e.g., polynomials)
  - Combining our approach with other techniques…

Questions?