Proof-Carrying Data
secure computation on untrusted execution platforms

Eran Tromer

Joint work with

Alessandro Chiesa

Ronald L. Rivest

CSAIL

Motivation

	CORRECTNESS	CONFINEMENT
SOFTWARE	• Bugs • Trojans	

- Software engineering (review, tests)
- Formal verification, static analysis
- Language type safety
- Dynamic analysis
- Reference monitors

Motivation

	CORRECTNESS	CONFINEMENT
SOFTWARE	• Bugs • Trojans	
NETWORK	• Lack of trust	

- Software engineering (review, tests)
- Formal verification, static analysis
- Language type safety
- Dynamic analysis
- Reference monitors

Motivation

	CORRECTNESS	CONFINEMENT
SOFTWARE	• Bugs • Trojans	
NETWORK	• Lack of trust	
PLATFORM	• Cosmic rays • Hardware bugs • Hardware trojans • IT supply chain	

- Software engineering (review, tests)
- Formal verification, static analysis
- Language type safety
- Dynamic analysis
- Reference monitors

Information technology supply chain: headlines

Recall previous talk by Dean Collins.

The New York Times (May 9, 2008)

"F.B.I. Says the Military Had Bogus Computer Gear"

ars technica (October 6, 2008)

"Chinese counterfeit chips causing military hardware crashes"

The New York Times (May 6, 2010)

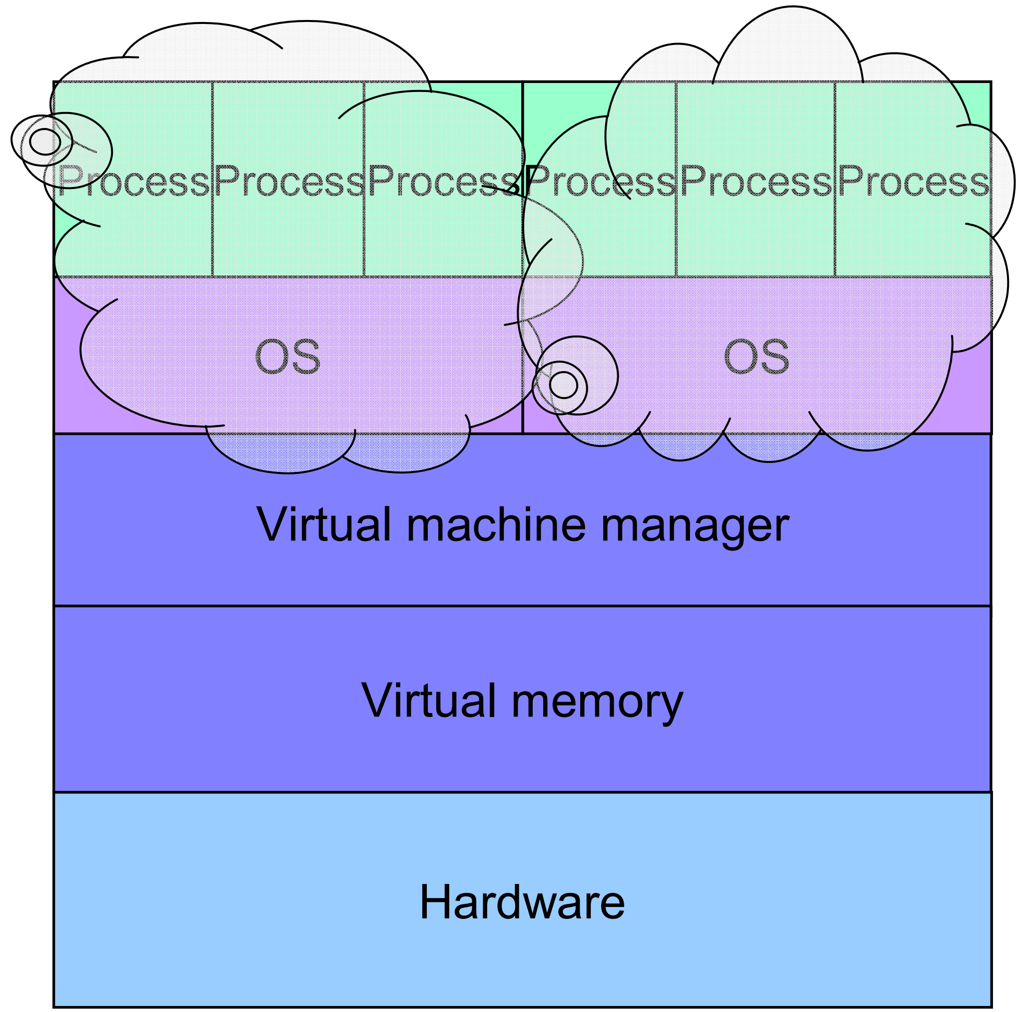
"A Saudi man was sentenced [...] to four years in prison for selling counterfeit computer parts to the Marine Corps for use in Iraq and Afghanistan."

Validation? Verification? Certification?

Motivation

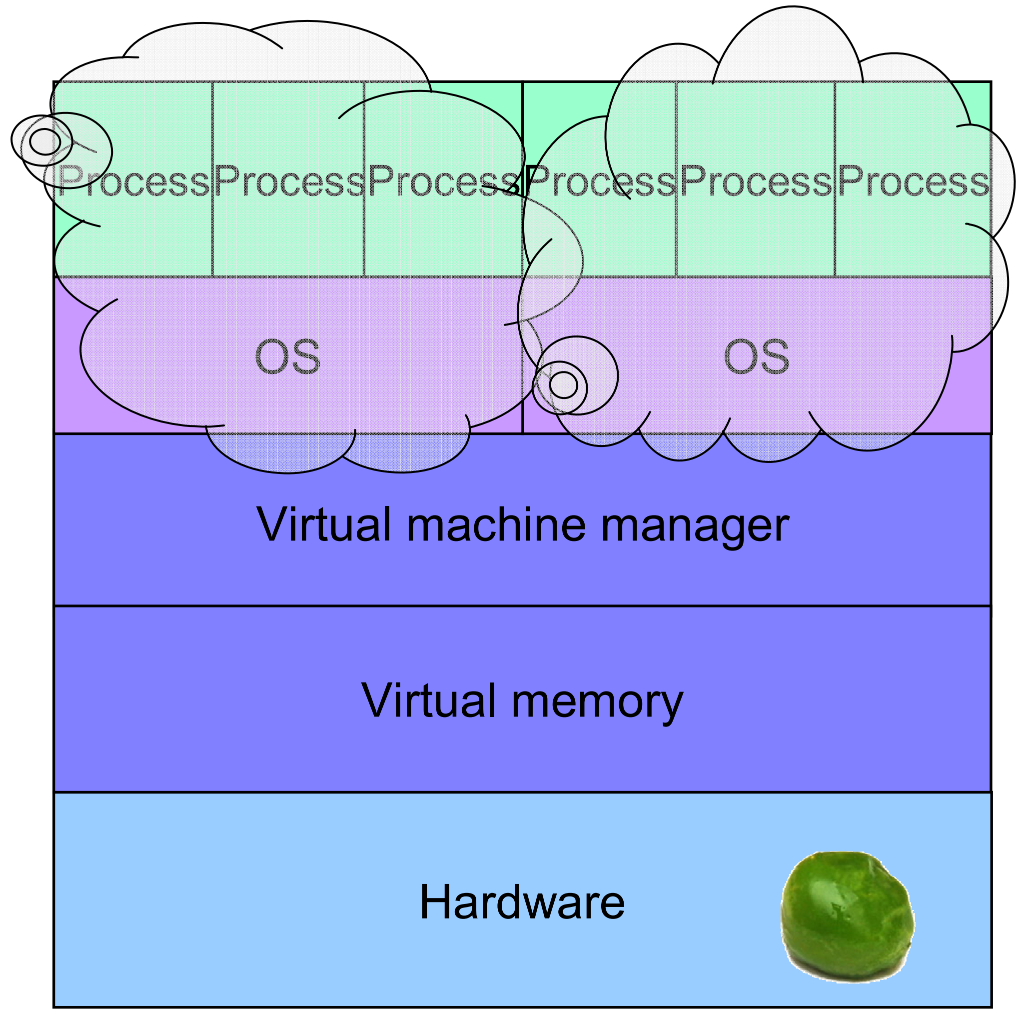
	CORRECTNESS	CONFINEMENT
SOFTWARE	• Bugs • Trojans	
NETWORK	• Lack of trust	
PLATFORM	• Cosmic rays • Hardware bugs • Hardware trojans • IT supply chain	→ Fault analysis • Architectural side-channels (e.g., cache attacks)

Example: cache attacks. Textbook virtualized architecture:



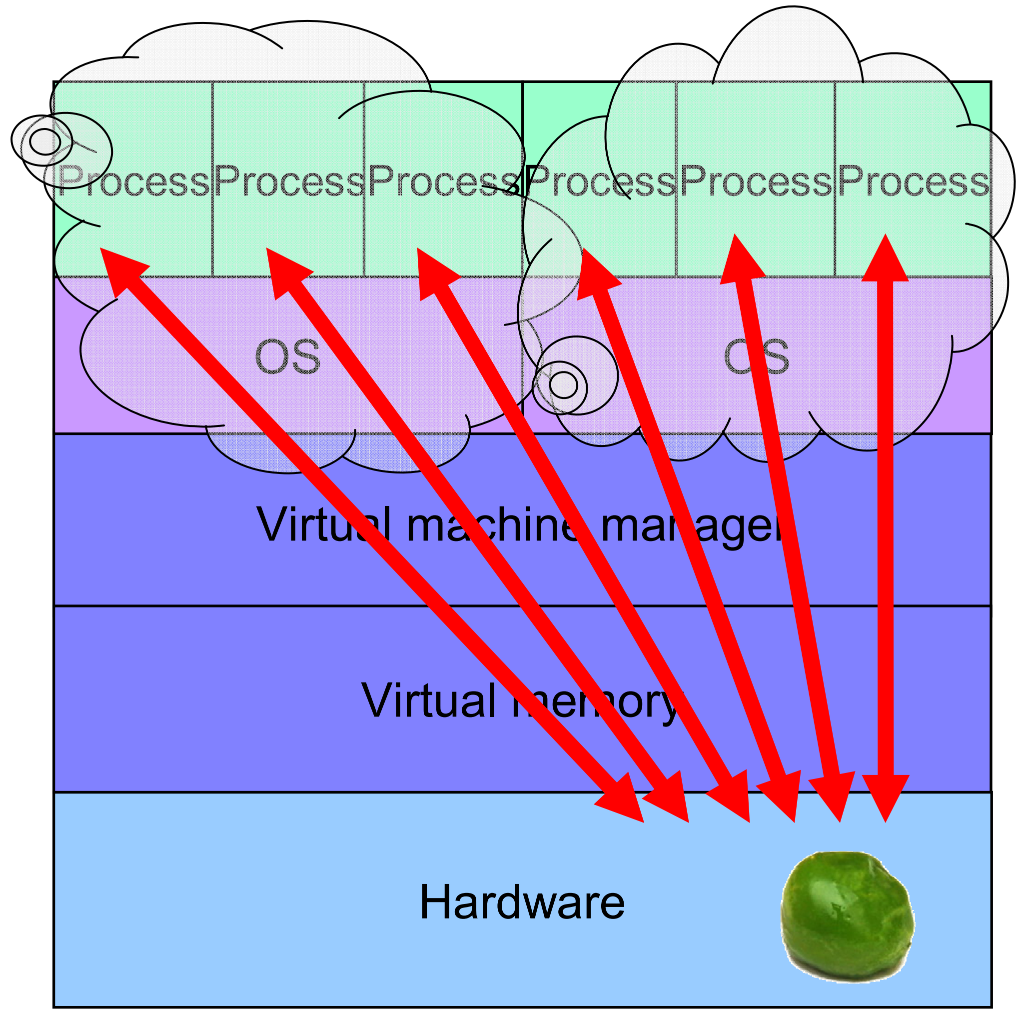
Another virtualized architecture:

Bedtime stories vs. architectural crosstalk



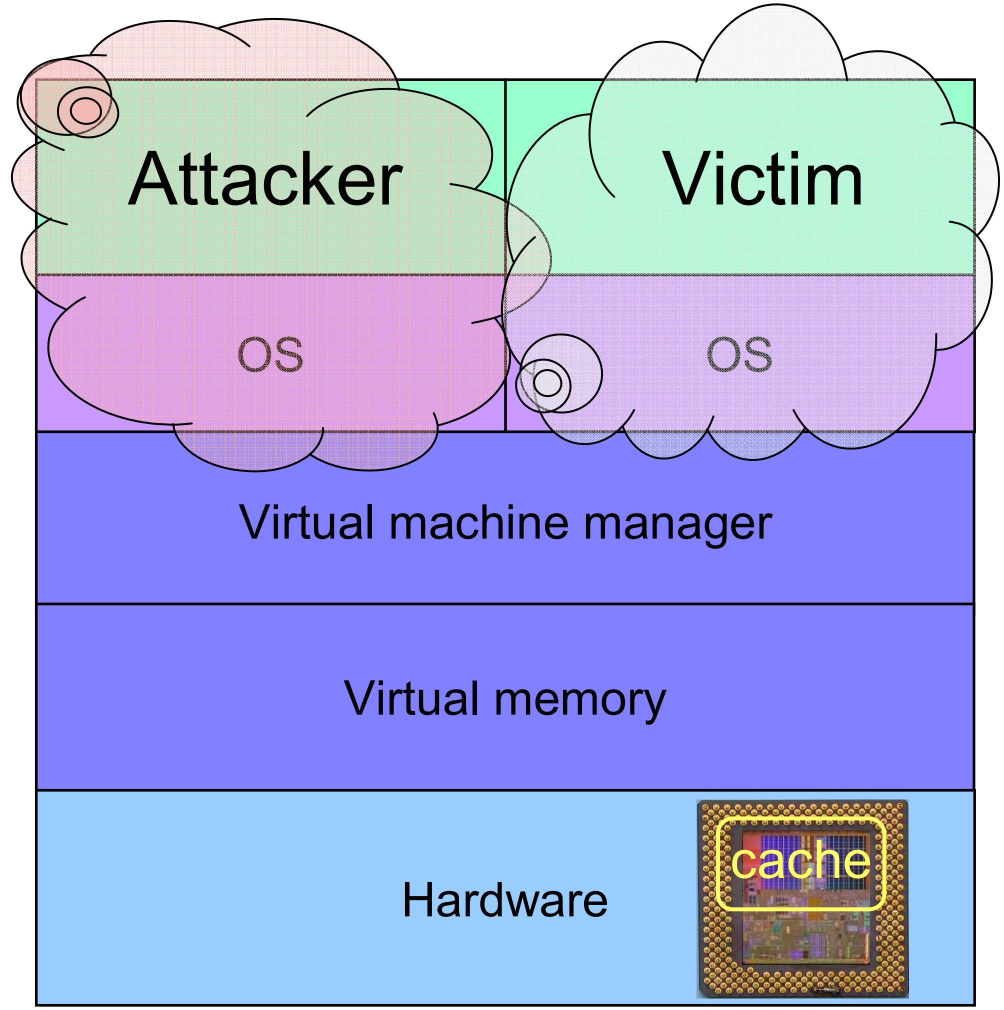
Cross-talk through architectural channels

- Contention for shared hardware resources



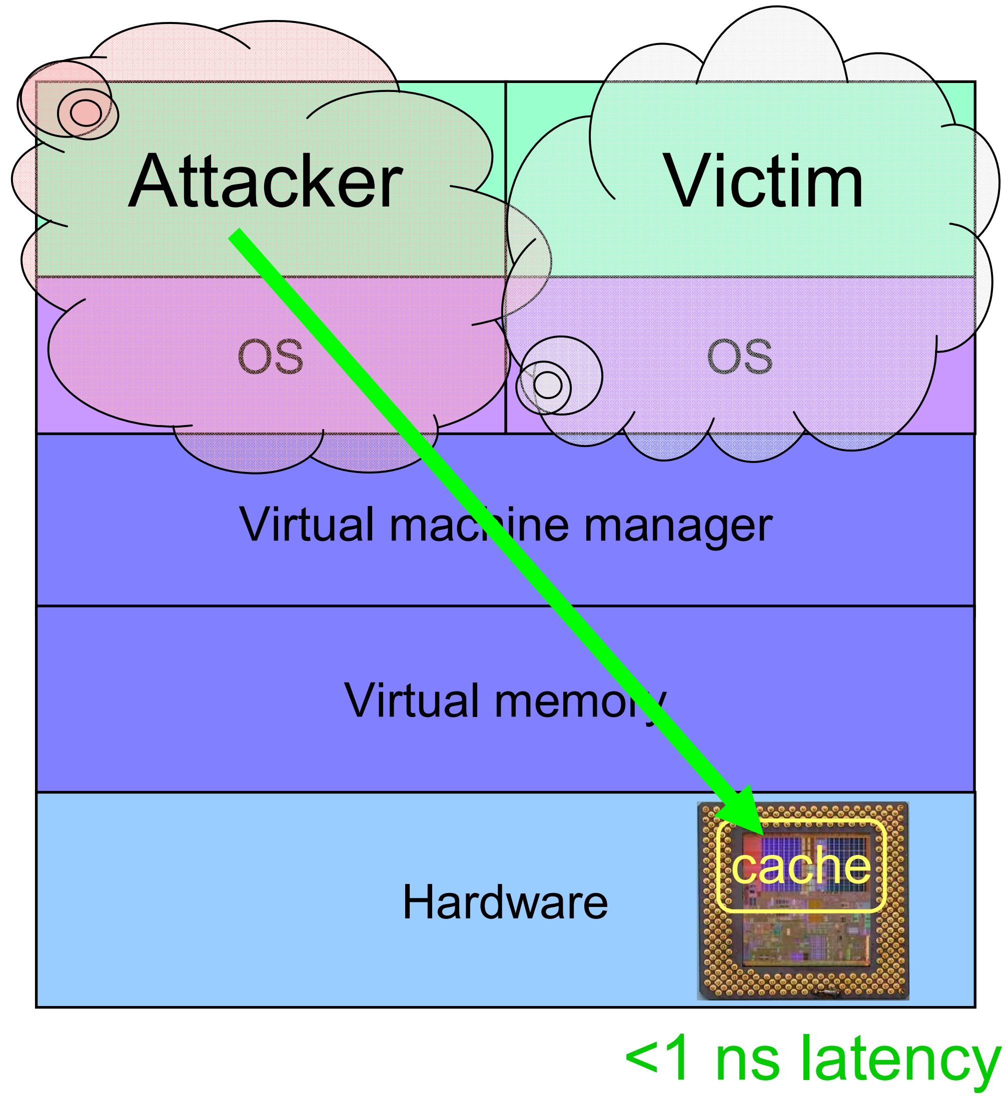
Cross-talk through architectural channels

- Contention for shared hardware resources
- Example: contention for **CPU data cache**



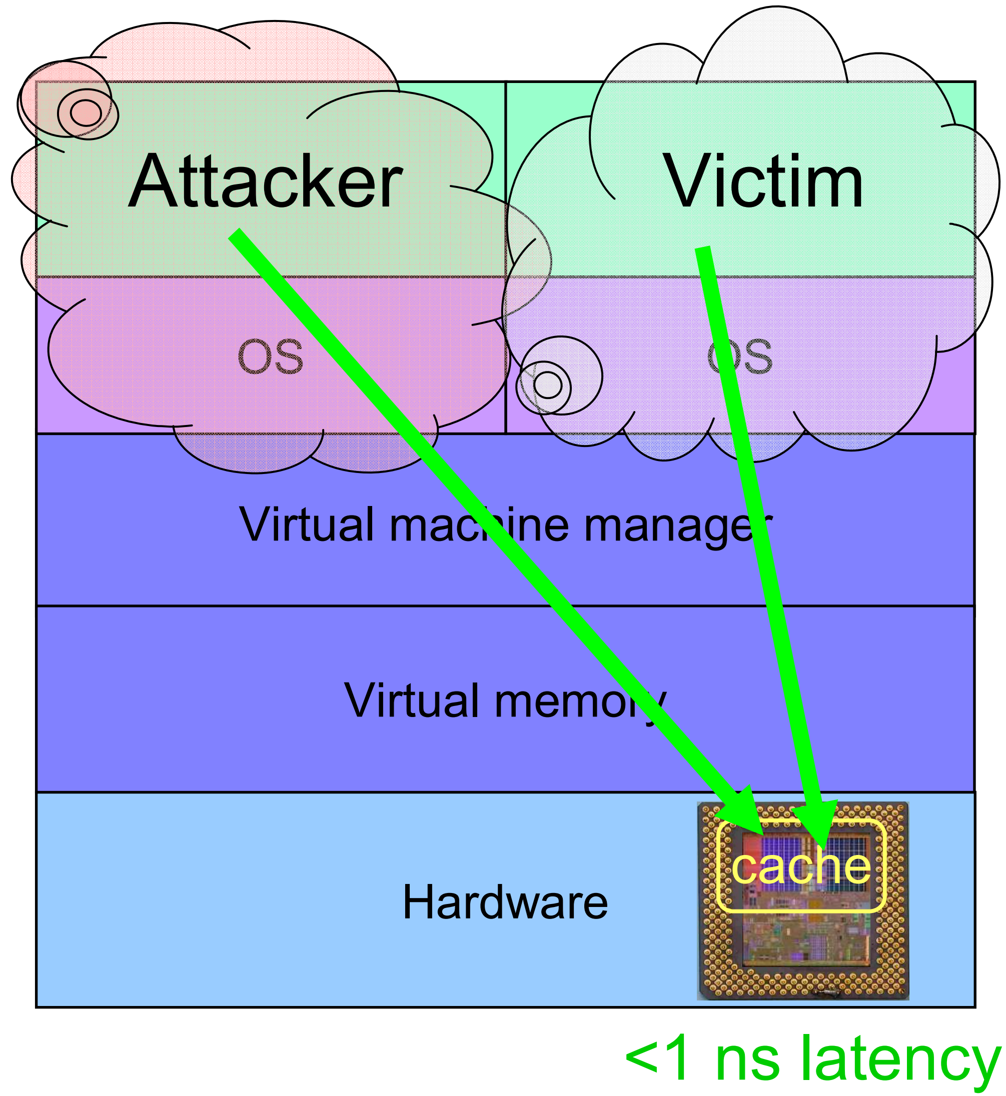
Cross-talk through architectural channels

- Contention for shared hardware resources
- Example: contention for **CPU data cache**



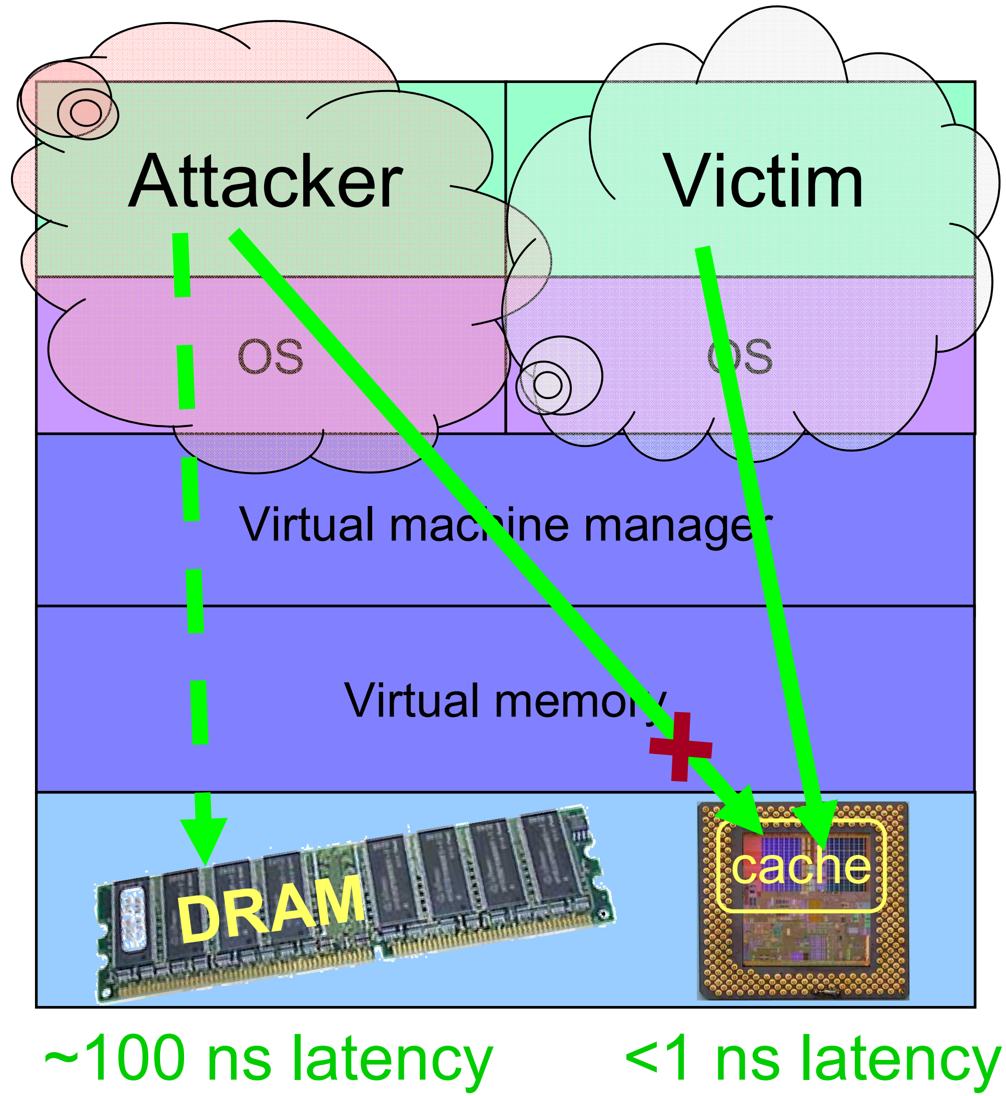
Cross-talk through architectural channels

- Contention for shared hardware resources
- Example: contention for **CPU data cache**



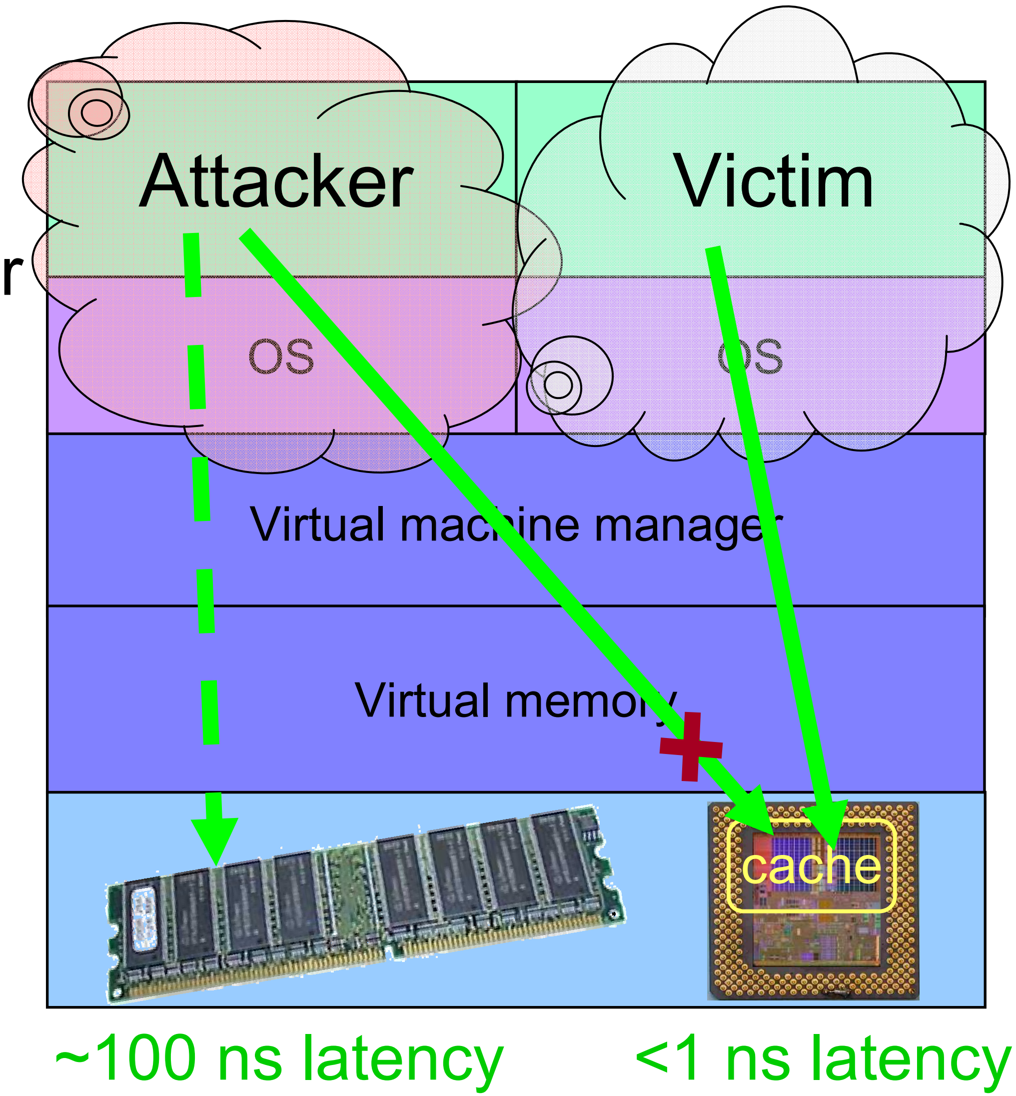
Cross-talk through architectural channels

- Contention for shared hardware resources
- Example: contention for **CPU data cache**



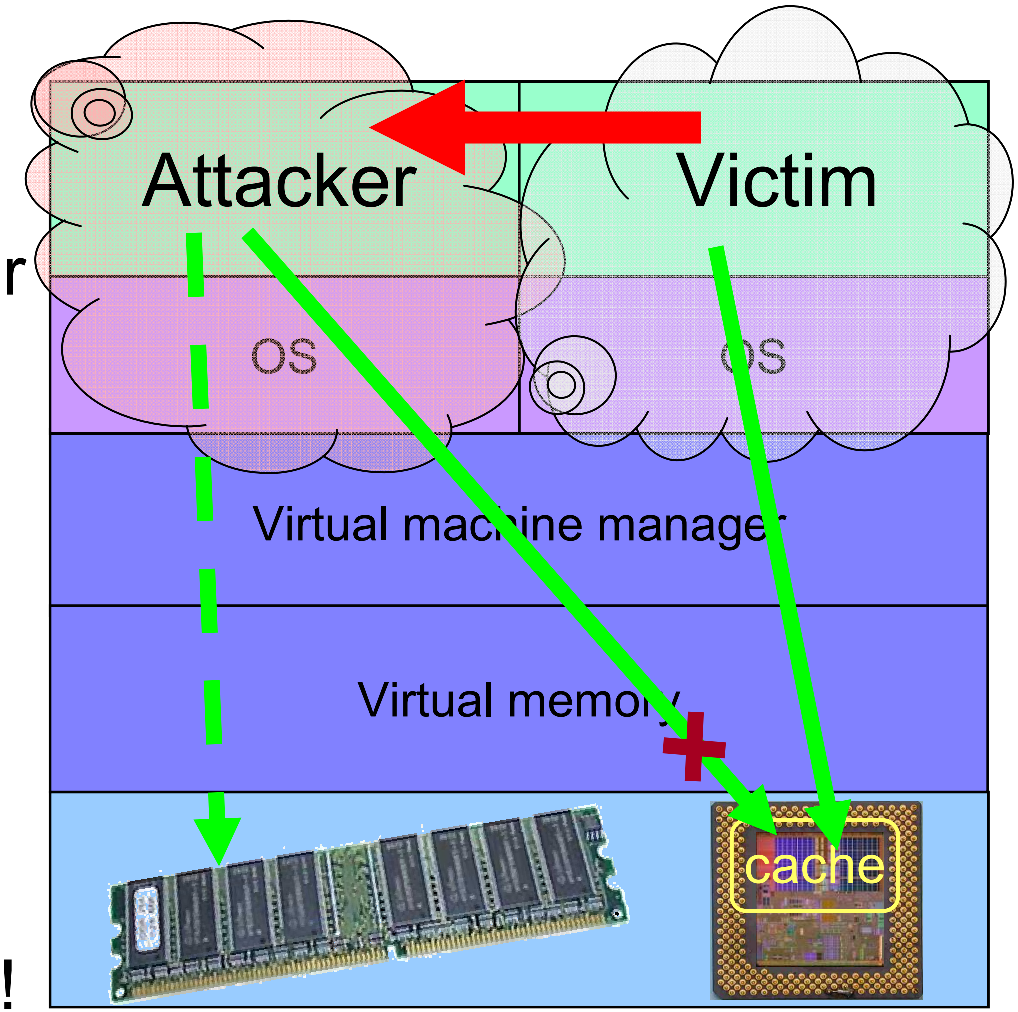
Cross-talk through architectural channels

- Contention for shared hardware resources
- Example: contention for CPU data cache leaks memory access patterns: addresses and timing.



Cross-talk through architectural channels

- Contention for shared hardware resources
- Example: contention for CPU data cache leaks memory access patterns: addresses and timing.
- The cached data is subject to memory protection…
- but even the metadata is sensitive information!

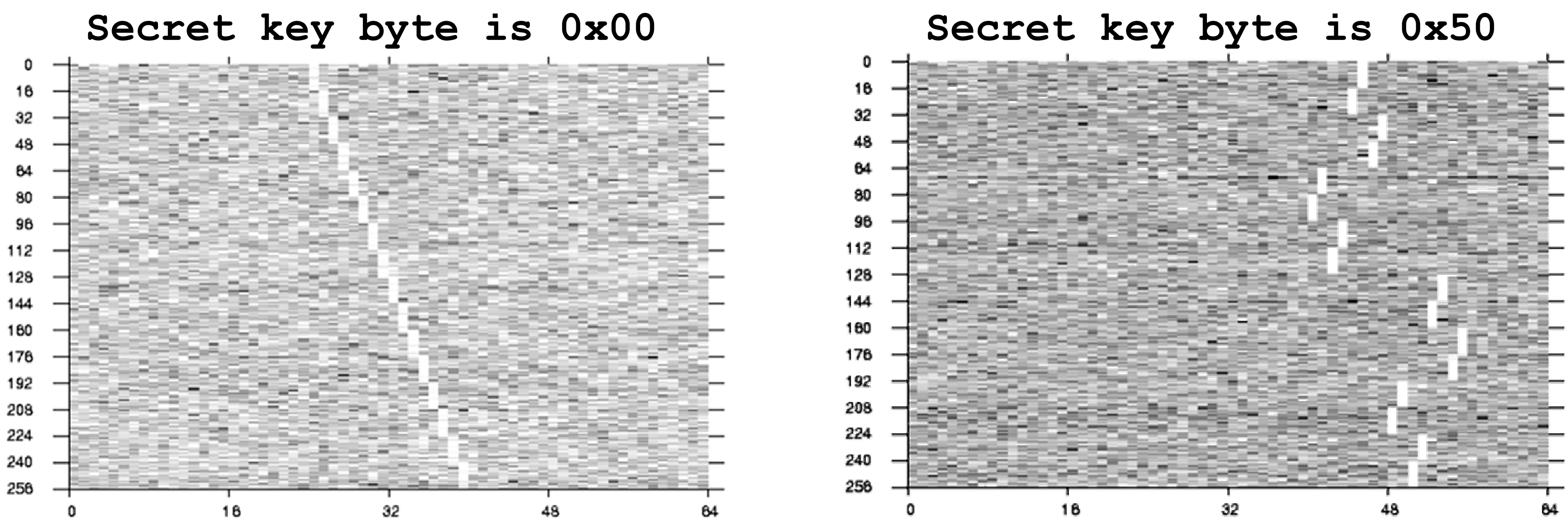


[Osvik Shamir Tromer 05]
[Tromer Osvik Shamir 09]

Stealing a disk encryption on a desktop:

(128-bit AES encryption, Linux **`dm-crypt`**)

Full key extracted from 65ms of measurements.



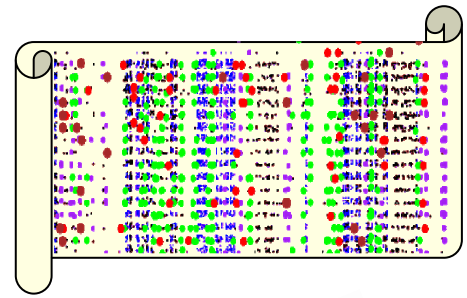
Measuring a "black box" OpenSSL encryption on Athlon 64, using 10,000 samples. Horizontal axis: evicted cache set. Vertical axis: `p[0]` (left), `p[5]` (right). Brightness: encryption time (normalized)

Information Leakage in Third-Party Compute Clouds

[Ristenpart Tromer Shacham Savage 09]

Demonstrated, using Amazon EC2 as a study case:

- Cloud cartography
  Mapping the structure of the "cloud" and locating a target on the map.



- Placement vulnerabilities
  An attacker can place his VM on the same physical machine as a target VM (40% success for a few dollars).

- Cross-VM exfiltration
  Once VMs are co-resident, information can be exfiltrated across VM boundary:
  - Covert channels
  - Load traffic analysis
  - Keystrokes

Motivation

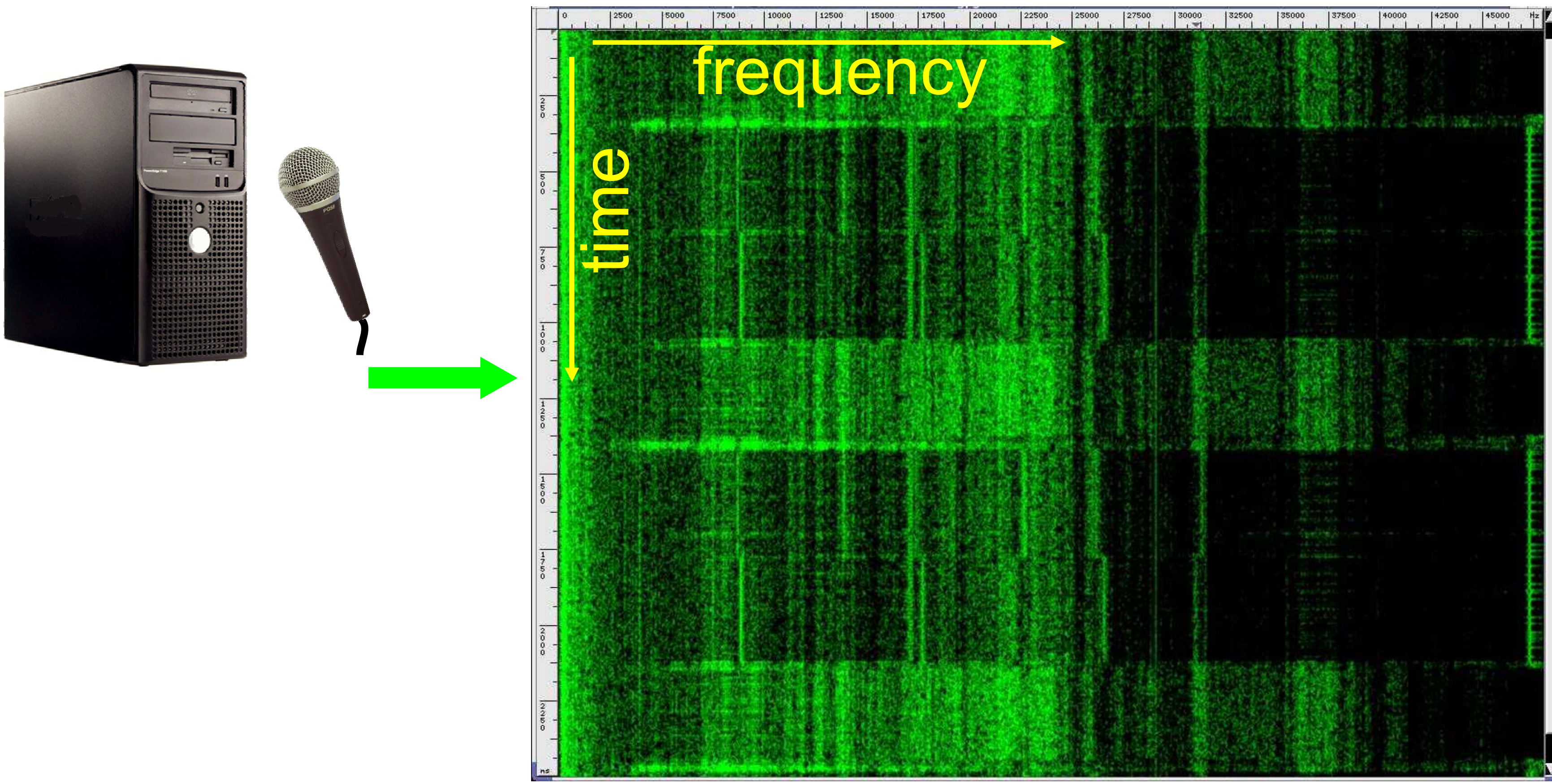
	CORRECTNESS	CONFINEMENT
SOFTWARE	• Bugs • Trojans	
NETWORK	• Lack of trust	
PLATFORM	• Cosmic rays • Hardware bugs • Hardware trojans • IT supply chain	Fault analysis • Architectural side-channels (e.g., cache)

Motivation

	CORRECTNESS	CONFINEMENT
SOFTWARE	• Bugs • Trojans	
NETWORK	• Lack of trust	
PLATFORM	• Cosmic rays • Hardware bugs • Hardware trojans • IT supply chain	Fault analysis • Architectural side-channels (e.g., cache)
ENVIRONMENT	• Tampering	• Physical side-channels (EM, power, acoustic)

Example: acoustic signatures of RSA signatures

[Shamir Tromer]



CSAIL MIT

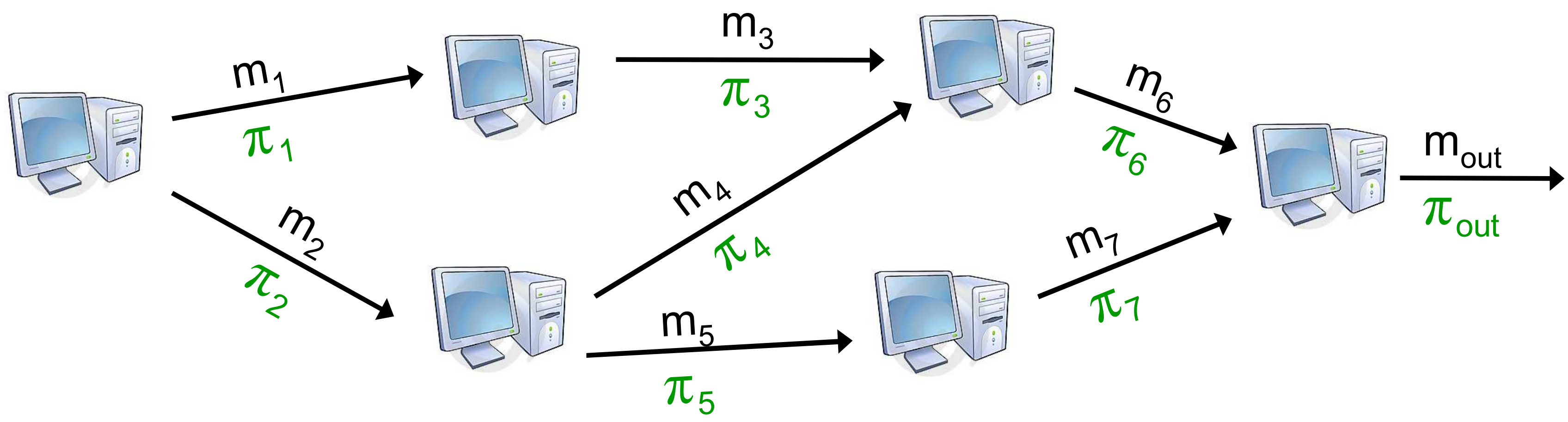
Motivation

	CORRECTNESS	CONFINEMENT
SOFTWARE	• Bugs • Trojans	
NETWORK	• Lack of trust	
PLATFORM	• Cosmic rays • Hardware bugs • Hardware trojans • IT supply chain	Fault analysis • Architectural side-channels (e.g., cache)
ENVIRONMENT	• Tampering	• Physical side-channels (EM, power, acoustic)

High-level goal

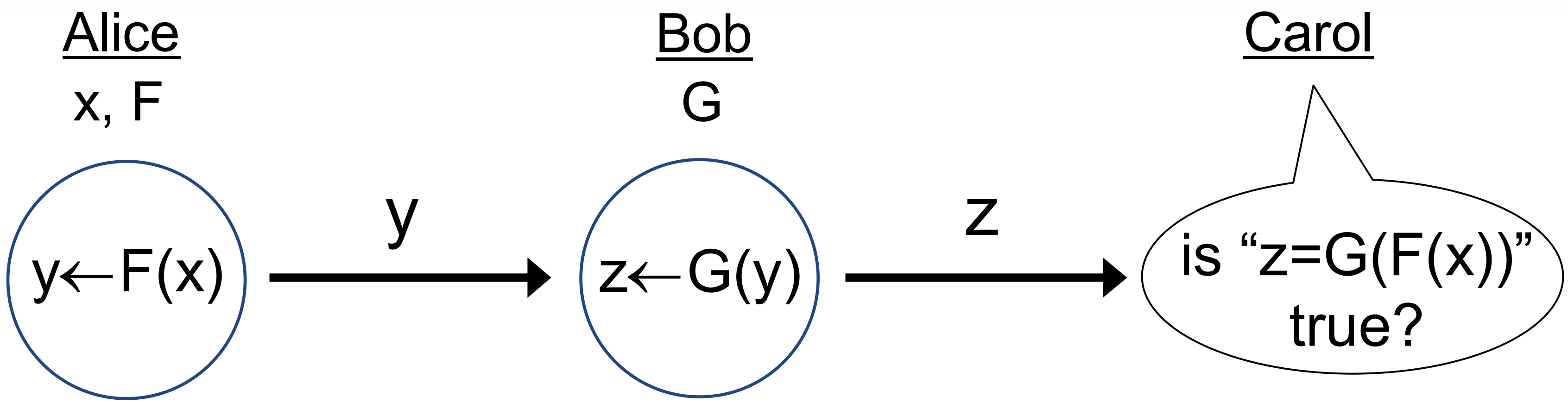
Ensure properties of a distributed computation when parties are mutually untrusting, faulty, leaky & malicious.

Approach: Proof-Carrying Data

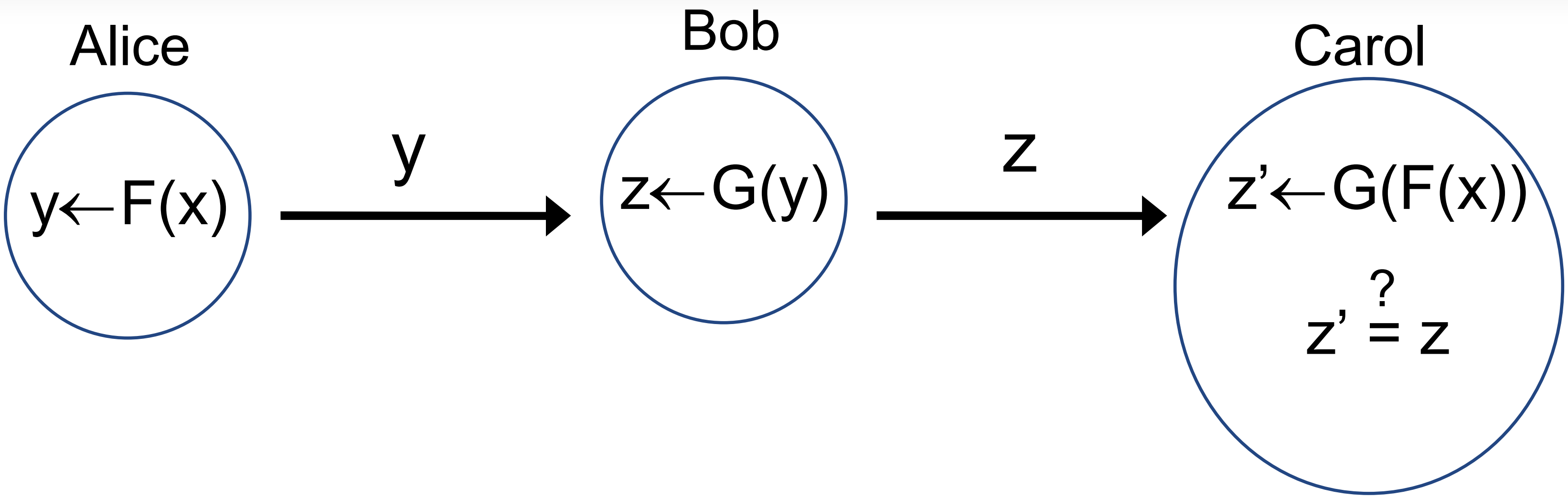


- Every message is augmented with a proof attesting to its "compliance".
- Compliance can express any property that can be verified by locally checking every node.
- Proofs can be verified efficiently and retroactively.

Toy example: 3-party correctness



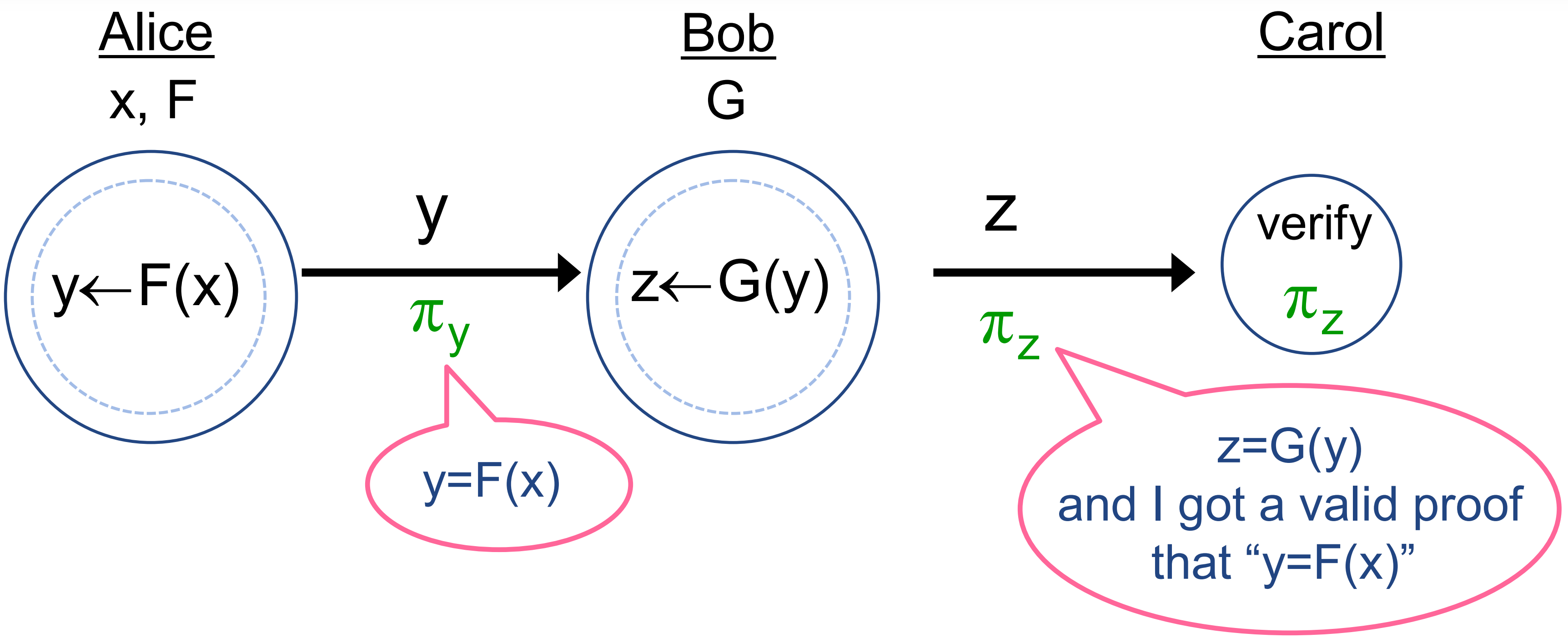
Example: trivial solution



Carol can recompute everything, but:

- Uselessly expensive
- Requires Carol to fully know x,F,G
  - We will want to represent these via short hashes/signatures

Example: Proof-Carrying Data [Chiesa Tromer 09]
following Incrementally-Verifiable Computation [Valiant 08]



Each party prepares a proof string for the next one.

Each proof is:

- Tiny (polylogarithmic in party's own computation).
- Efficiently verifiable by the next party.

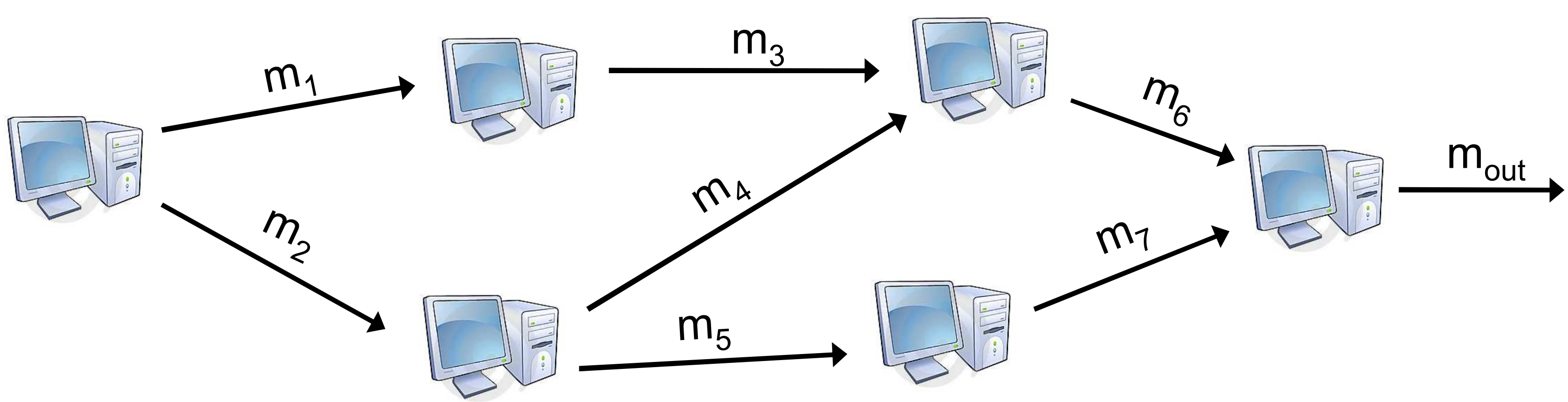
Generalizing:

The Proof-Carrying Data framework

Generalizing: distributed computations

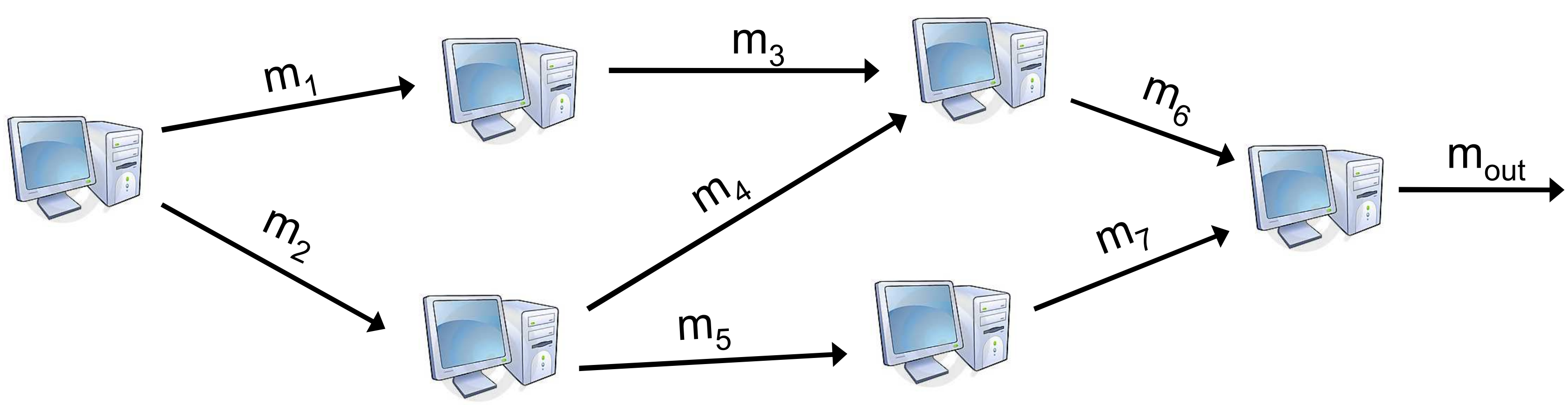
Distributed computation:

Parties exchange messages and perform computation.



Generalizing: arbitrary interactions

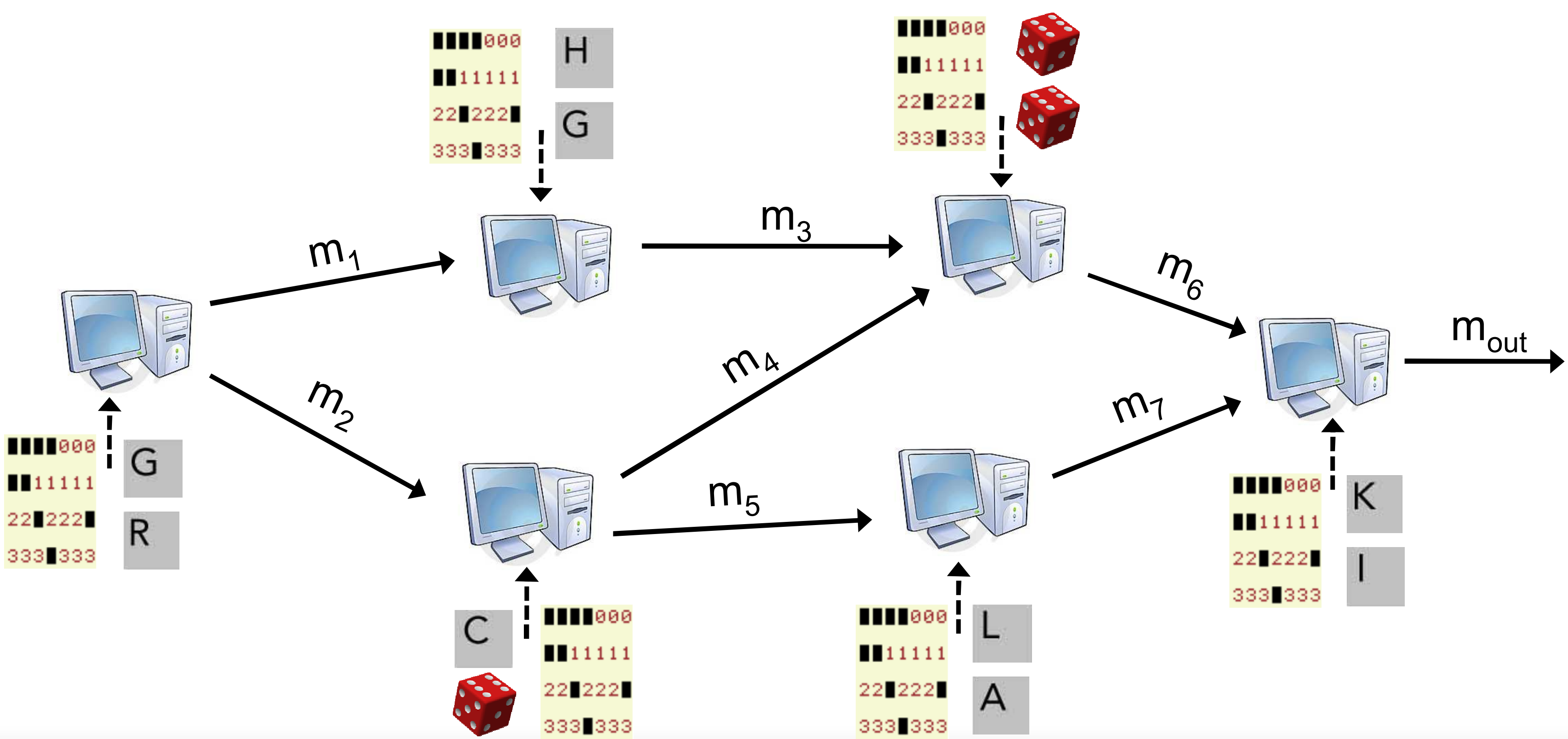
- Arbitrary interactions
  - communication graph over time is any DAG



Generalizing: arbitrary interactions

- Computation and graph are determined on the fly
  - by each party's local inputs:

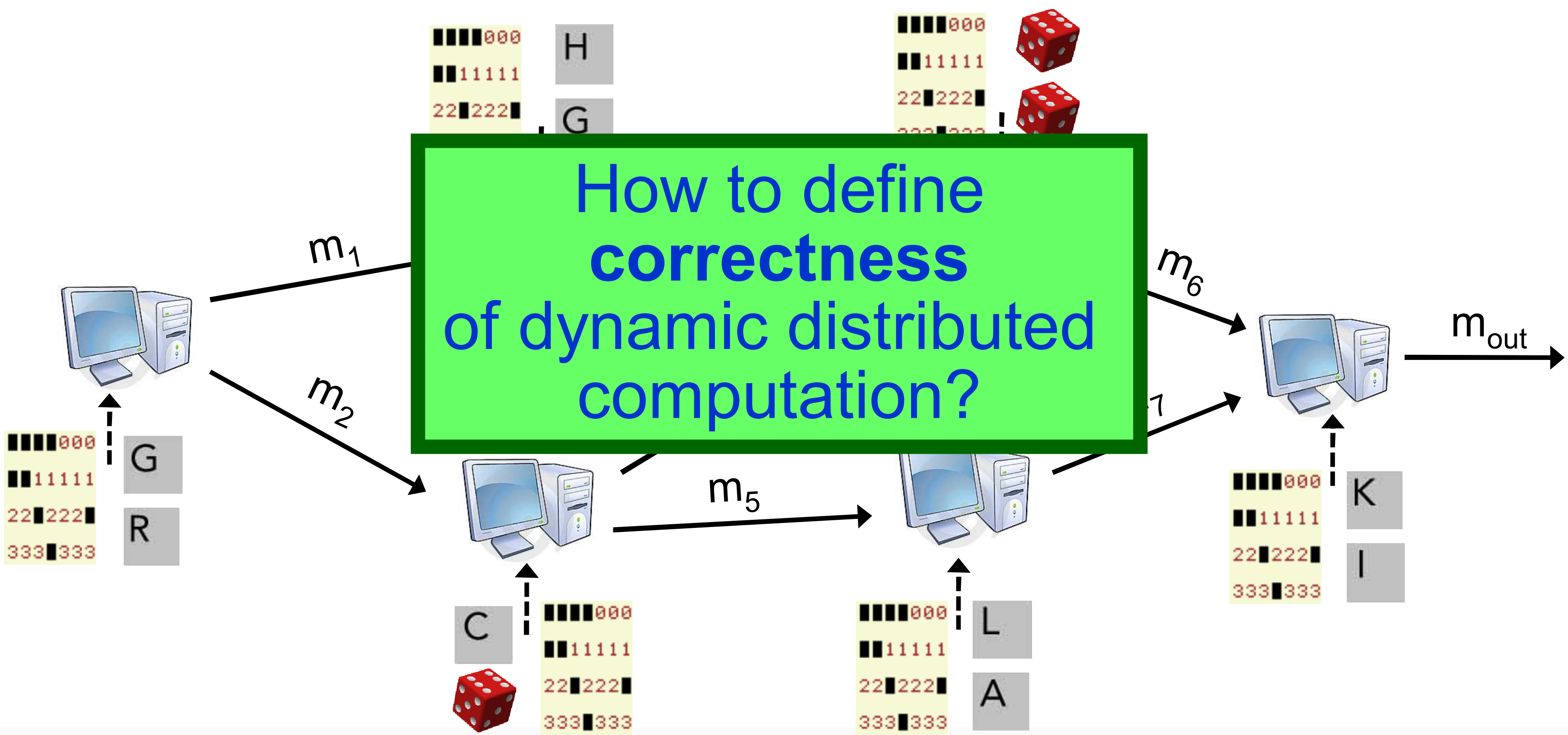
human inputs randomness program



Generalizing: arbitrary interactions

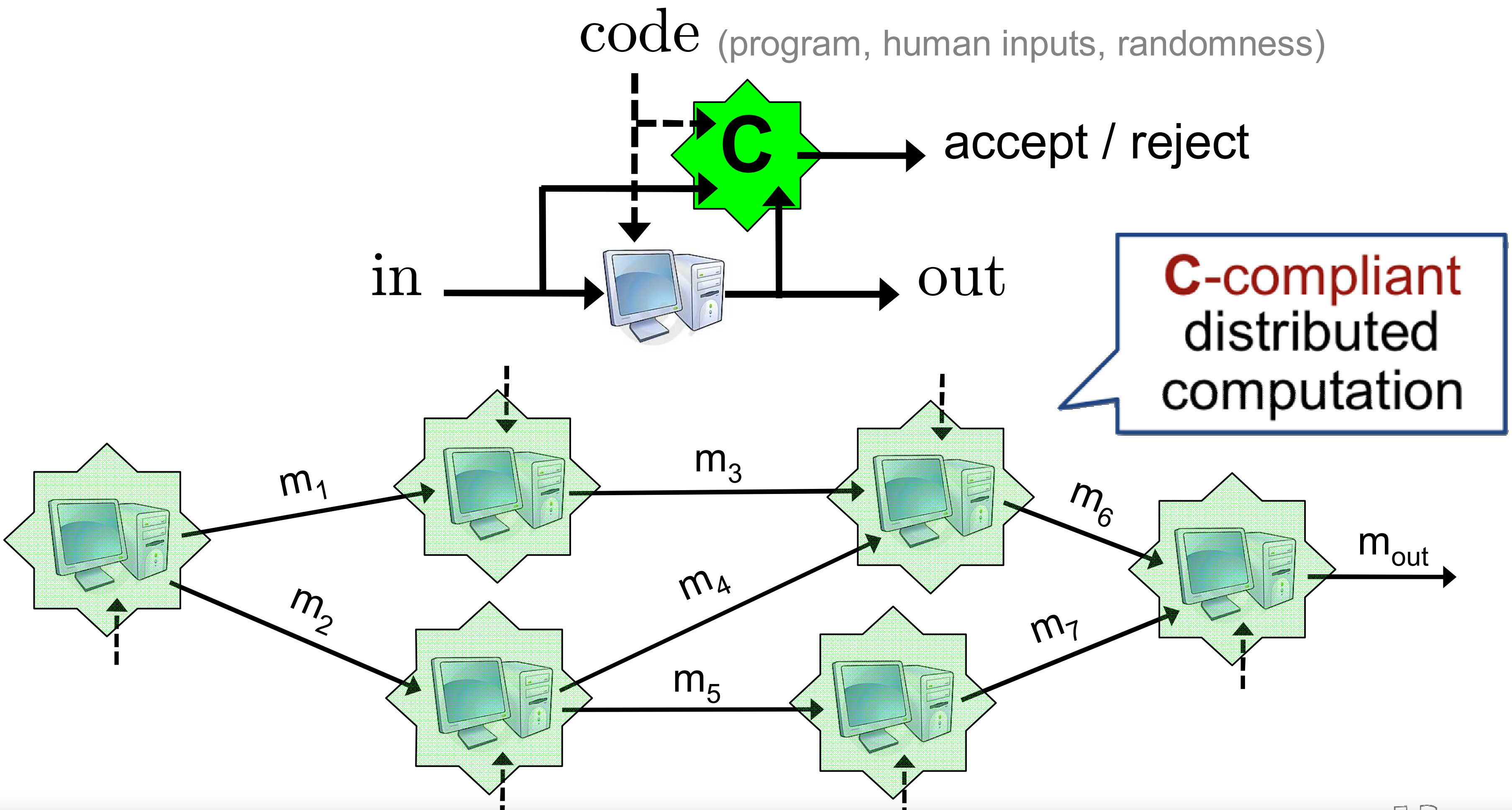
- Computation and graph are determined on the fly
  - by each party's local inputs:

human inputs randomness program



C-compliance

System designer specifies his notion of **correctness** via a **compliance predicate** **C**$(\mathrm{in},\mathrm{code},\mathrm{out})$ that must be locally fulfilled at every node.

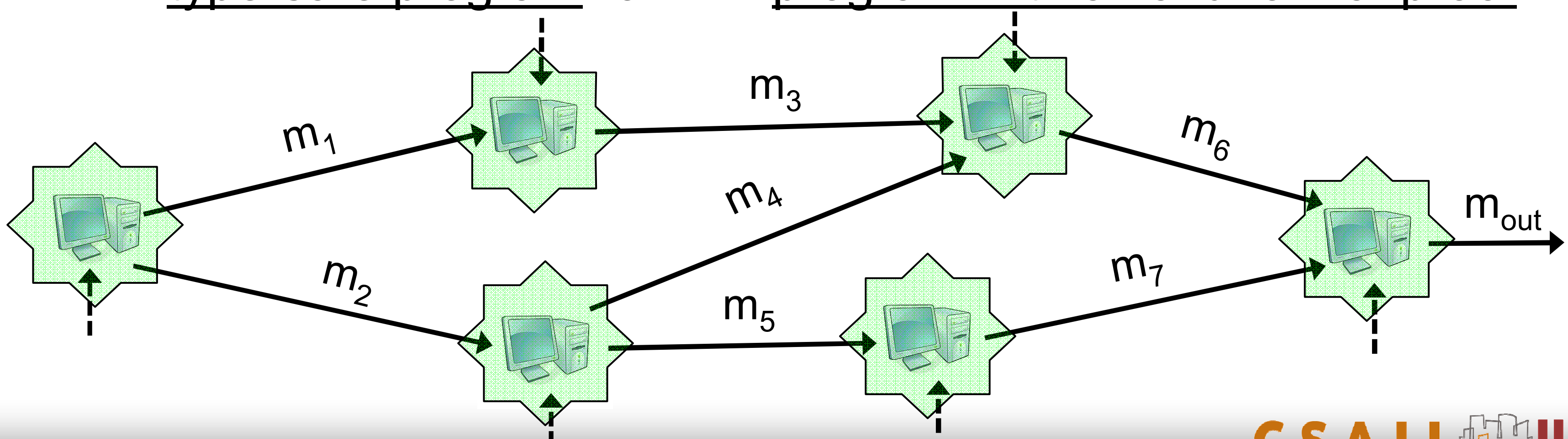


Examples of **C**-compliance

correctness is a **compliance predicate** **C**(in,code,out) that must be locally fulfilled at every node

Some examples:

- **C** = "the output is the result of correctly computing a <u>prescribed</u> program"
- **C** = "the output is the result of correctly executing some program <u>signed by the sysadmin</u>"
- **C** = "the output is the result of correctly executing some <u>type-safe program</u>" or "… "<u>program with a valid formal proof</u>"



Goals

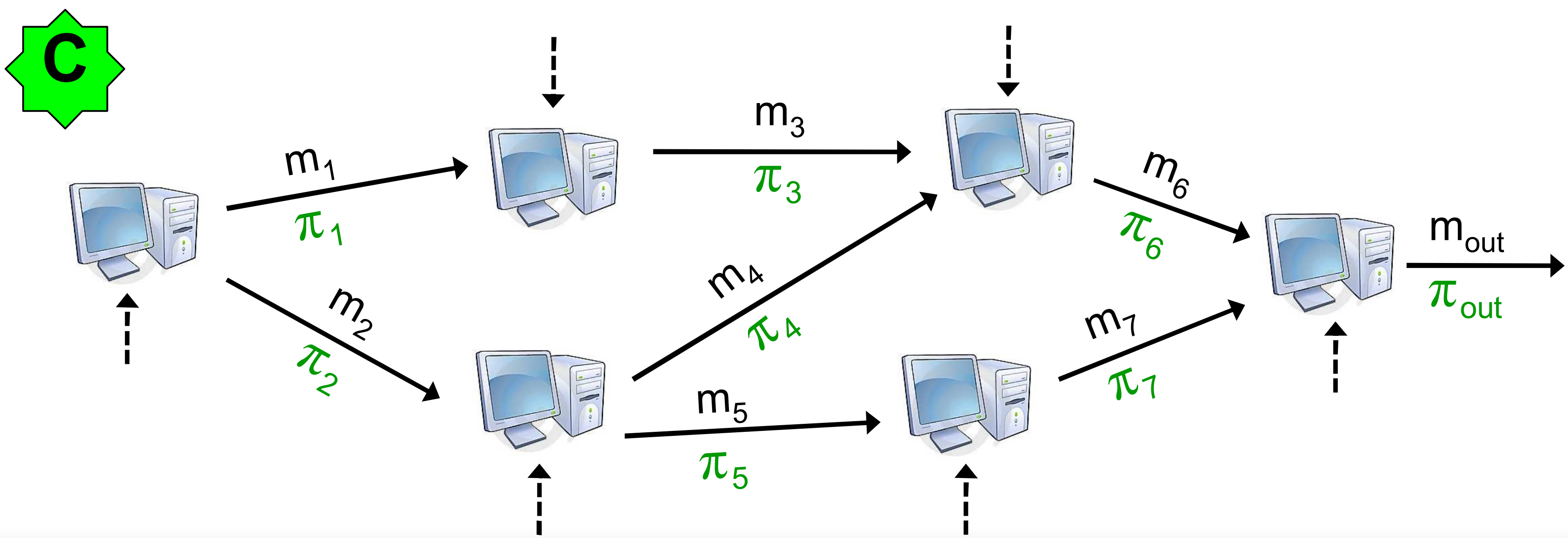
Ensure **C**-compliance while **respecting** the original distributed computation.

- Allow for any interaction between parties
- Preserve parties' communication graph
  - no new channels
- Allow for dynamic computations
  - human inputs, indeterminism, programs
- Blowup in computation and communication is local and polynomial

Dynamically augment computation with proofs strings

In PCD, messages sent between parties are augmented with concise proof strings attesting to their "compliance".

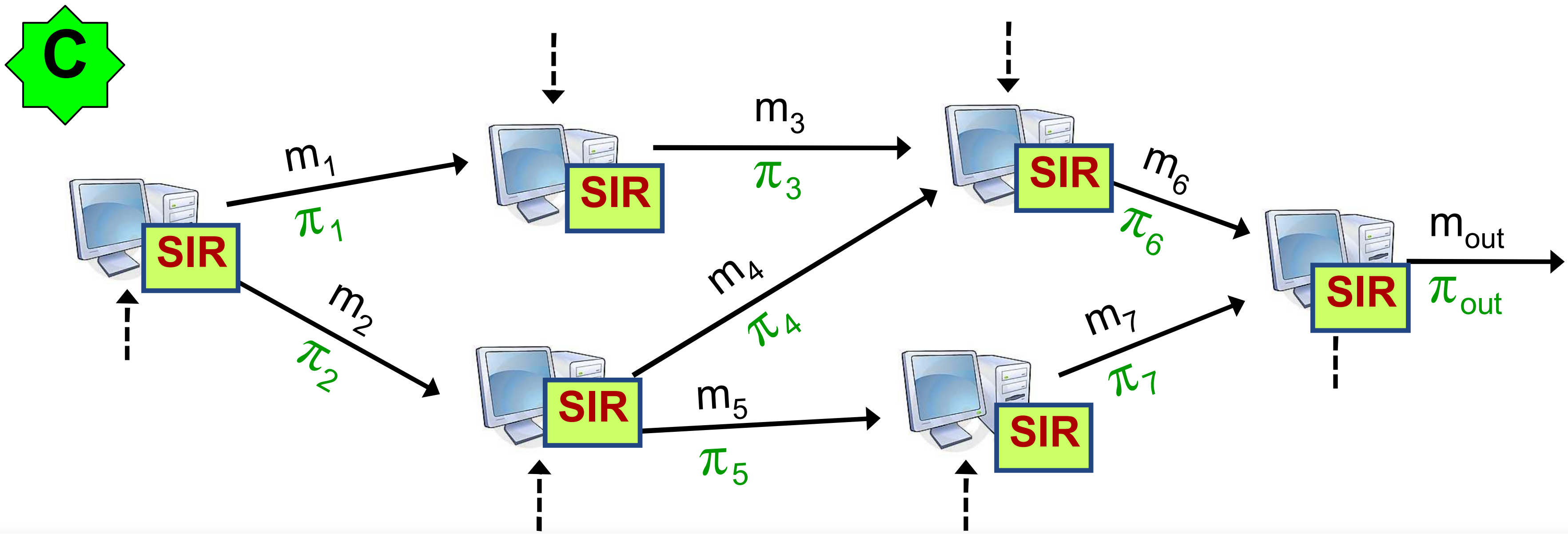
Distributed computation evolves like before, except that each party also generates on the fly a proof string to attach to each output message.



Model

Every node has access to a simple, fixed, stateless trusted functionality -- essentially, a signature card.

- Signed-Input-and-Randomness (SIR) oracle



(Some) envisioned applications

Correctness and integrity of IT supply chain

- Consider a system as a collection of components, with specified functionalities
  - Chips on a motherboard
  - Servers in a datacenter
  - Software modules
- **C**$(\mathrm{in},\mathrm{code},\mathrm{out})$ checks if the component's specification holds
- Proofs are attached across component boundaries
- If a proof fails, computation is locally aborted

  $\rightarrow$ integrity, attribution

Application: type safety

C$(\mathrm{in},\mathrm{code},\mathrm{out})$ verifies that
code is type-safe & $\mathrm{out}=\mathrm{code}(\mathrm{in})$

- Using PCD, type safety can be maintained
  - even if underlying execution platform is untrusted
  - even across mutually untrusting platforms

- Type safety is very expressive
  - Can express any computable property
  - Extensive literature on types that can be verified efficiently (at least with heuristic completeness – good enough!)

Multilevel security through Information Flow Control

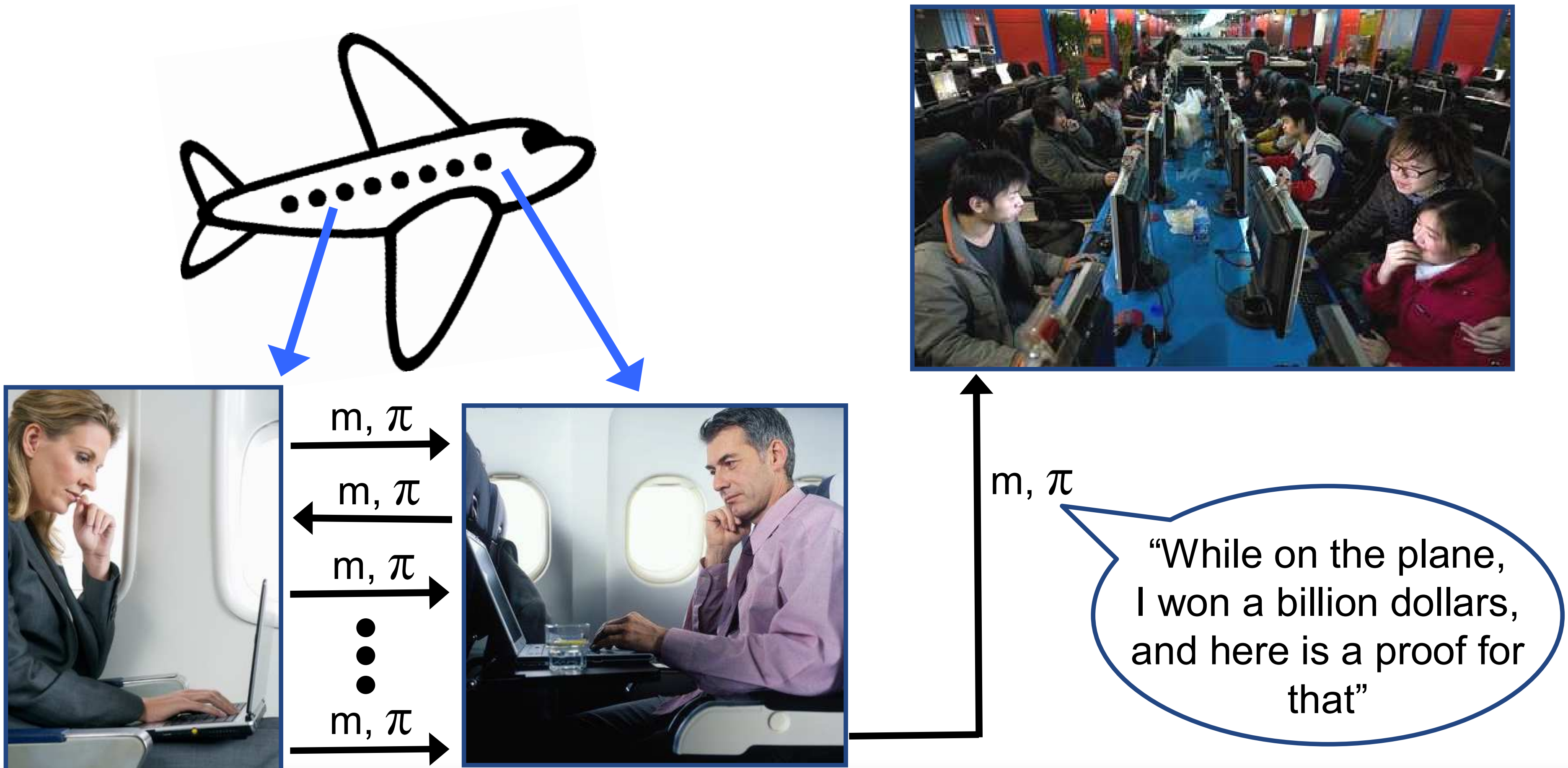
- Computation gets "`secret`" and "`non-secret`" inputs
- "`non-secret`" inputs are signed as such
- Any output labeled "`non-secret`" must be independent of `secret`s
- Use **C** to allow only computation on non-secret inputs, according to a fixed schedule.
  - Initial inputs must be signed
  - Subsequent computation respect Information Flow Control rules and follow fixed schedule
- Censor at system's perimeter:
  - Verifies proof on every outgoing message
  - Lets out only non-secret data.

Simulations and MMO

- Distributed simulation:
  - Physical models
  - Virtual worlds (massively multiplayer online virtual reality)
- How can participants prove they have "obeyed the laws of physics"?
  (e.g., cannot reach through wall into bank safe)
- Traditional: centralized.
- P2P architectures strongly motivated but insecure
  [Plummer '04] [GauthierDickey et al. '04]
- Use **C**-compliance to enforce the laws of physics.

Simulations and MMO: example

- Alice and Bob playing on an airplane, can later rejoin a larger group of players, and prove they did not cheat while offline.



Our results

Our results

- Formally define Proof Carrying Data
  - System administrator defines the compliance predicate
  - Existing software is mechanically translated to add proof generation
  - Compliance is automagically guaranteed
- Show a theoretical construction
  - “Polynomial time” – not yet practically feasible
  - Requires signature cards

Vision

PCD →

- type safety
- fault isolation & accountability
- multilevel security
- financial systems
- distributed dynamic program analysis
- proof-carrying code
- antispam email policies

⋮

Security design reduces to "**compliance engineering**": write down a suitable compliance predicate **C**.

Proof-Carrying Data: Conclusions and open problems

Contributions

- Framework for securing distributed computations between parties that are mutually untrusting and potentially faulty, leaky, and malicious
- Explicit construction, under standard generic assumptions, in a “signature cards” model
- Suggested applications

Ongoing and future work

- Achieve practicality (“polynomial time” PCP is not good enough!)
- Reduce requirement for signature cards, or prove necessity
- Add zero-knowledge constructions
- Identify and realize applications

Thanks!

THIS MORNING:

2 HOURS LATER: