Using System-Enforced Determinism to Control Timing Channels

Bryan Ford, Amittai Aviram, Weiyi Wu,
Jose Faleiro, Ramki Gummadi
Yale University
http://dedis.cs.yale.edu/

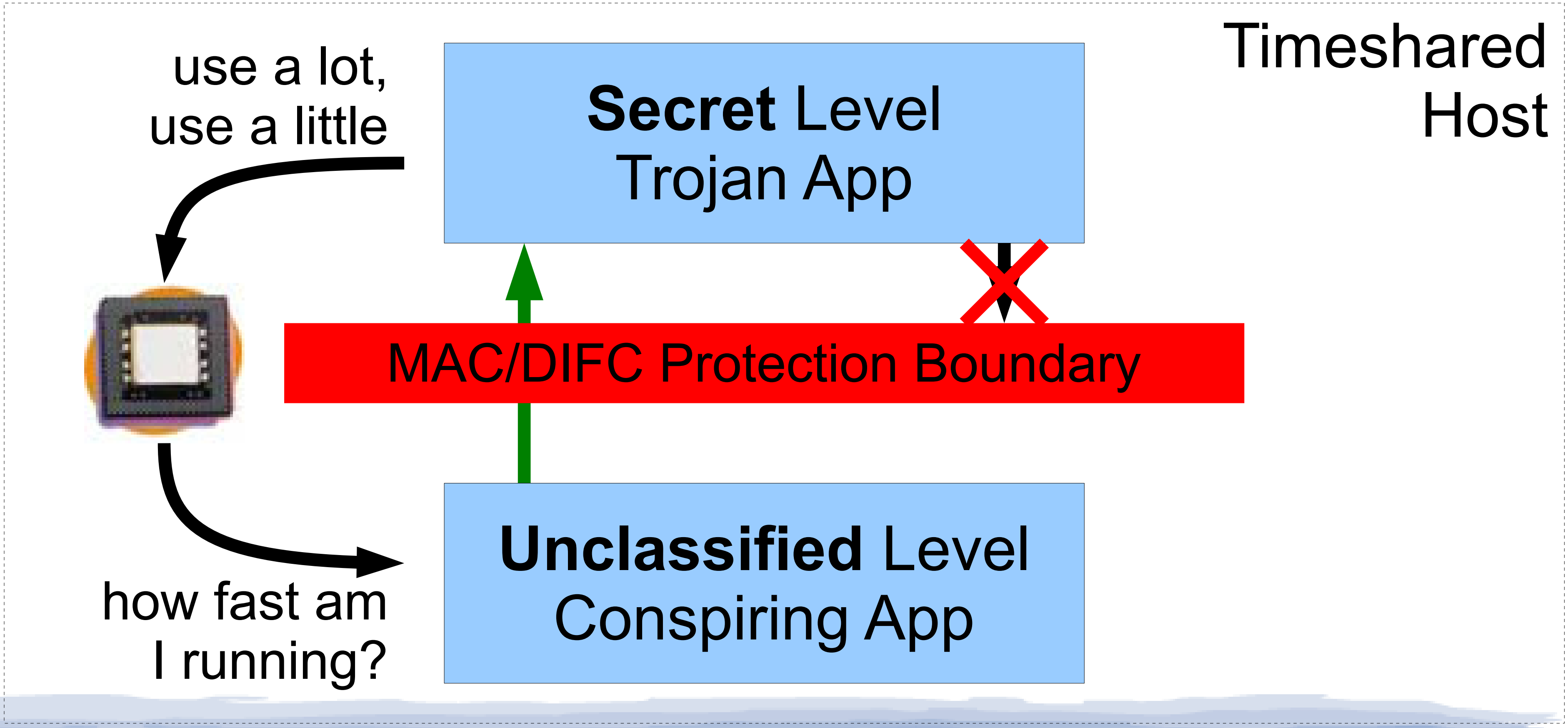
ASPLOS PC Symposium, November 2, 2012

The Long History of Timing Attacks

- **Cooperative attacks** – apply to:
  - Mandatory Access Control (MAC) systems [Kemmerer 83, Wray 91]
  - Decentralized Information Flow Control (DIFC) [Efstathopoulos 05, Zeldovich 06]
- **Non-cooperative attacks** – apply to:
  - Processes/VMs sharing a CPU core [Percival 05, Wang 06, Acıiçmez 07, …]
  - Including VM configurations typical of clouds [Ristenpart 09]

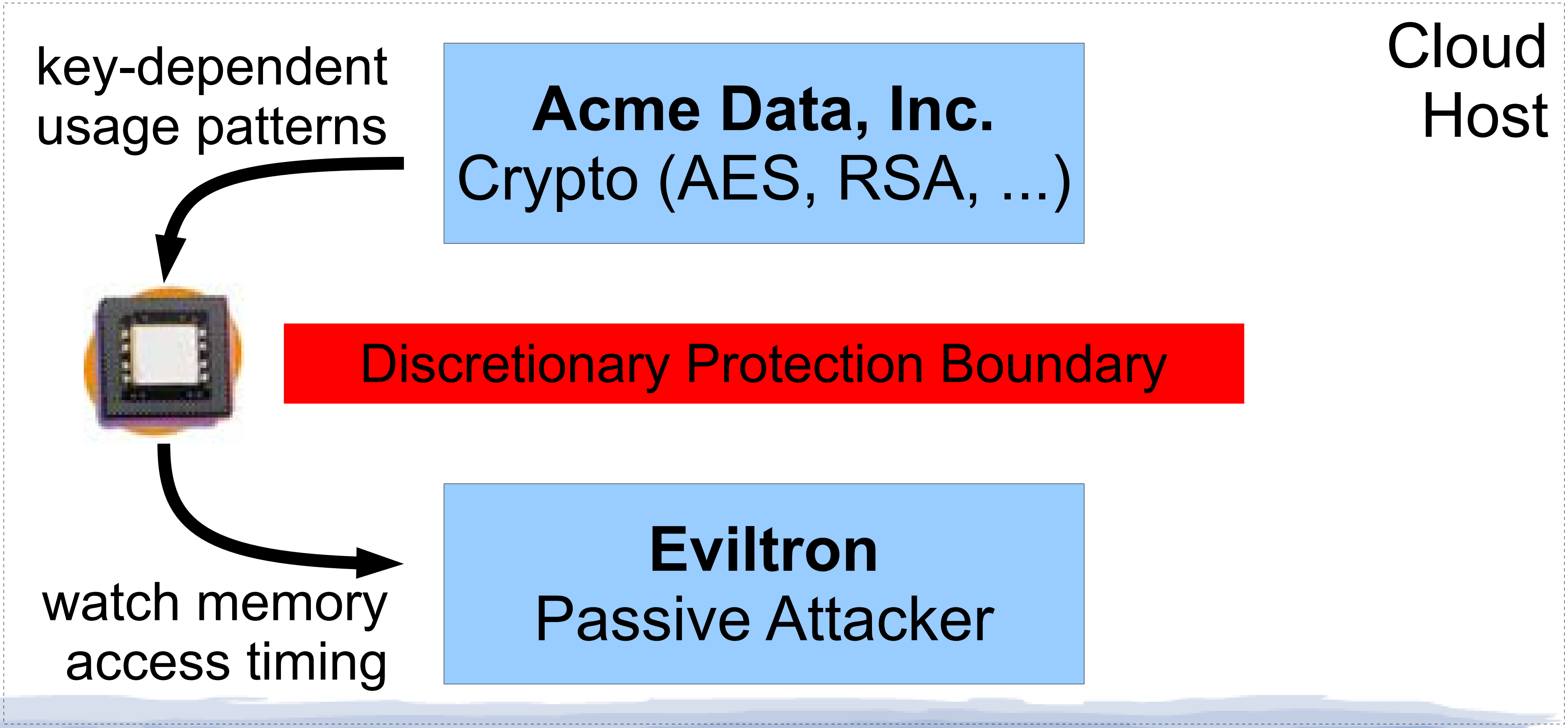
Cooperative Attacks: Example

Trojan leaks **secret** information by modulating a *timing channel* observable by **unclassified** app


Timeshared Host
use a lot, use a little
Secret Level Trojan App
MAC/DIFC Protection Boundary
how fast am I running?
Unclassified Level Conspiring App

Non-Cooperative Attacks: Example

Apps *unintentionally* modulate shared resources to reveal secrets when running standard code



Cloud Host

key-dependent usage patterns

Acme Data, Inc.
Crypto (AES, RSA, ...)

Discretionary Protection Boundary

watch memory access timing

Eviltron
Passive Attacker

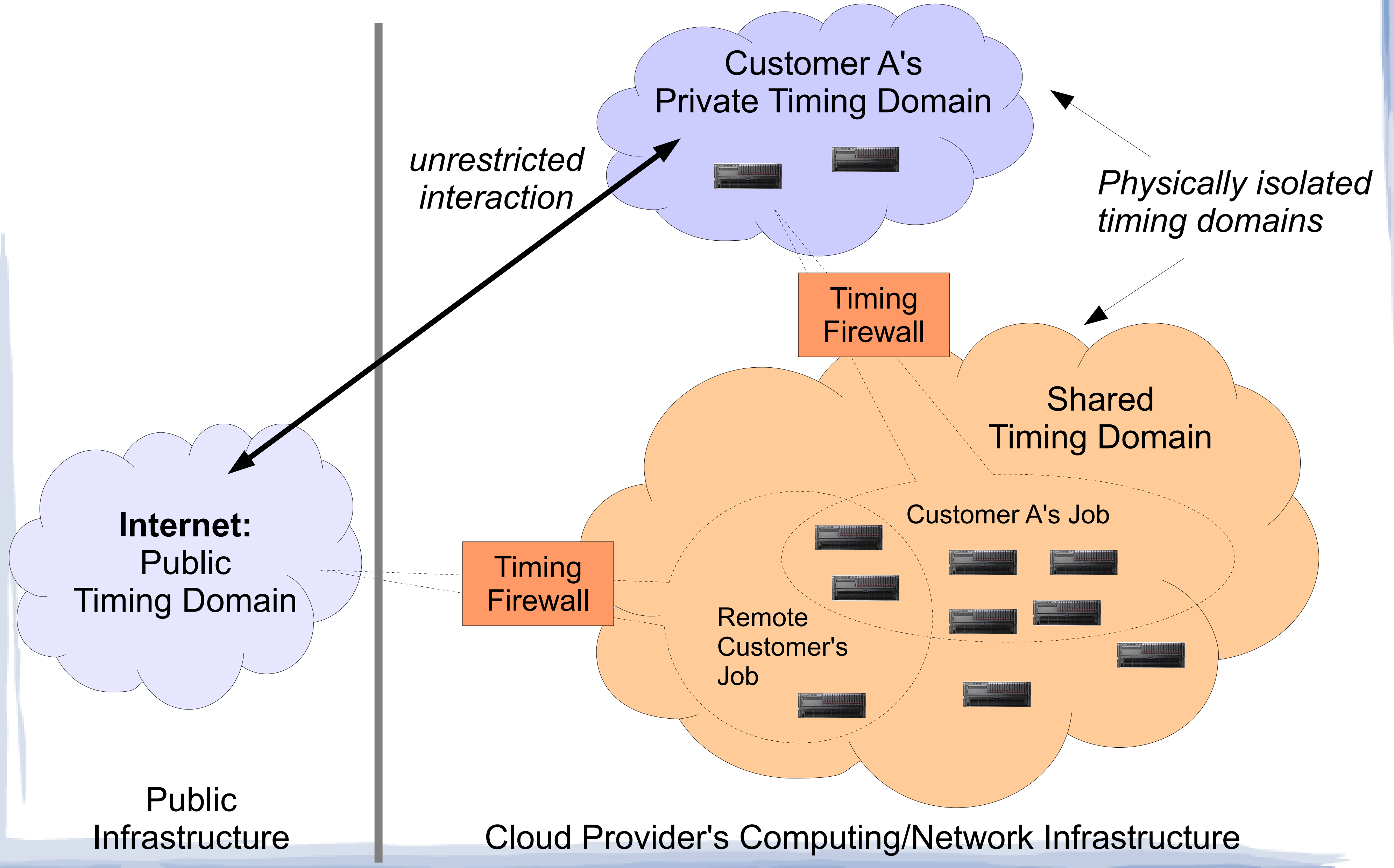
Timing Attacks in the Cloud

The cloud *exacerbates* timing channel risks:

1. Routine co-residency
2. Massive parallelism
3. No intrusion alarms → hard to monitor/detect
4. Partitioning defenses defeat elasticity

“*Determinating Timing Channels in Compute Clouds*”
[CCSW '10]

Towards a “Timing-Hardened Cloud”



Customer A's Private Timing Domain

unrestricted interaction

Physically isolated timing domains

Timing Firewall

Shared Timing Domain

Internet: Public Timing Domain

Timing Firewall

Customer A's Job

Remote Customer's Job

Public Infrastructure

Cloud Provider's Computing/Network Infrastructure

Leak-Plugging Approaches

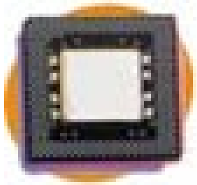
Two broad classes of existing solutions:

- *Tweak specific algorithms, implementations*
  - Equalize AES path lengths, cache footprint, …
- *Demand-insensitive resource partitioning*
  - Requires *new or modified hardware* in general
    - Partition CPU cores, cache, interconnect, …
  - Can't oversubscribe, stat-mux resources
    - → Not economically feasible in an “elastic” cloud!

Anatomy of a Timing Channel

Two elements required: [Wray 91]

- A *resource* that can be *modulated* by the signaling process (or victim)



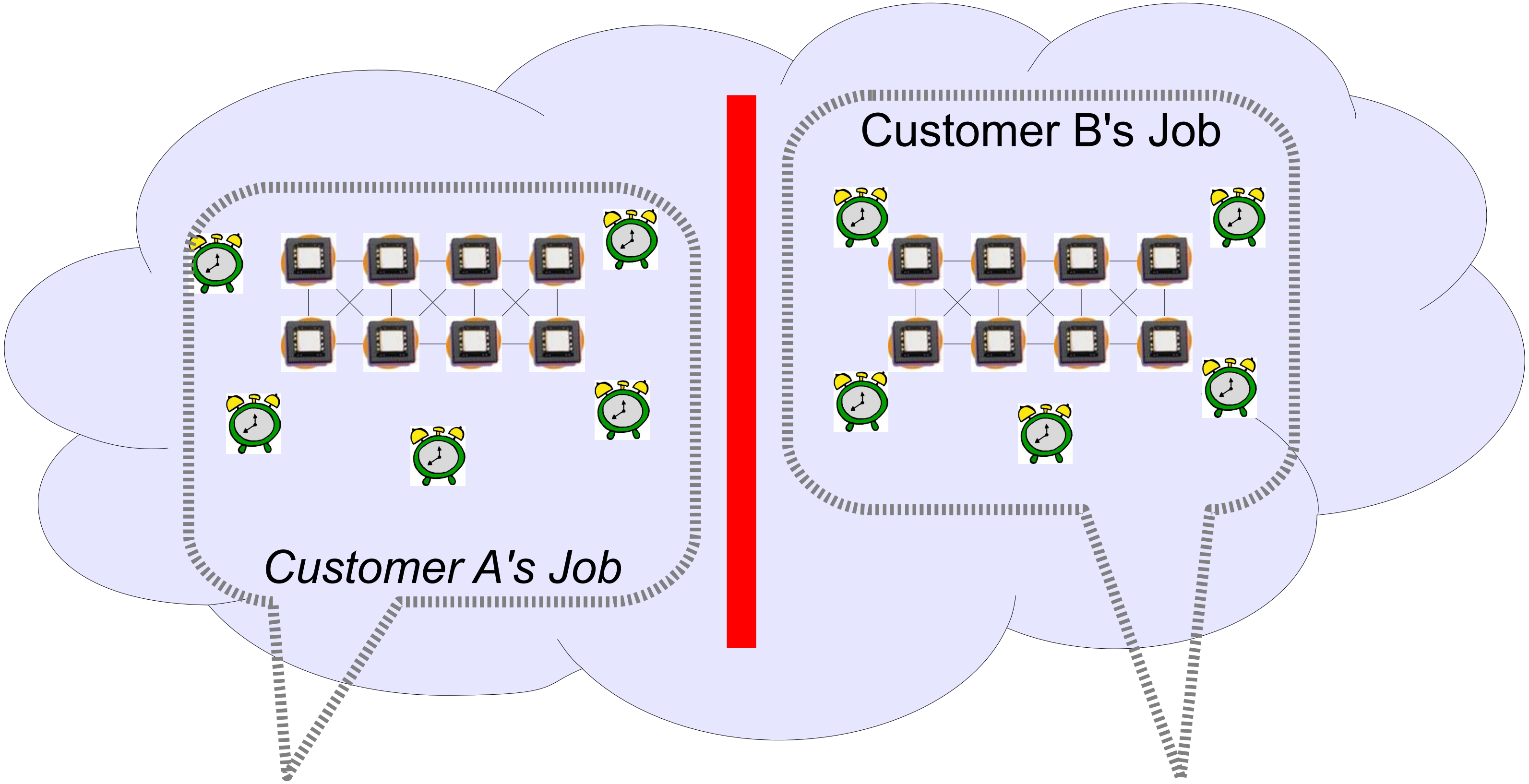
- A *reference clock* enabling the attacker to observe, extract the modulated signal

Remove either → no timing channel.

Prior Approaches

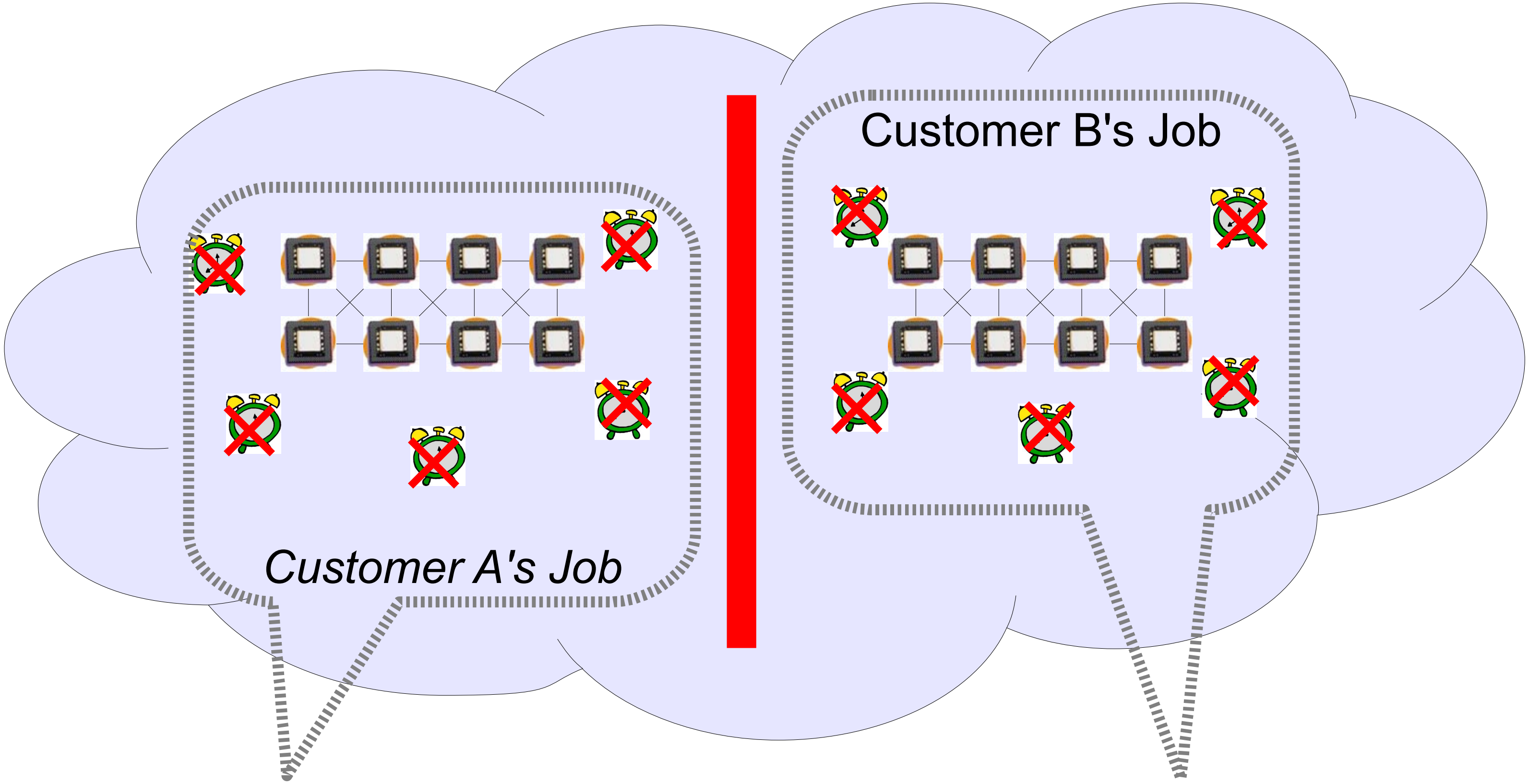
Attempt to **eliminate modulation**

- e.g., by partitioning hardware resources



Our Approach

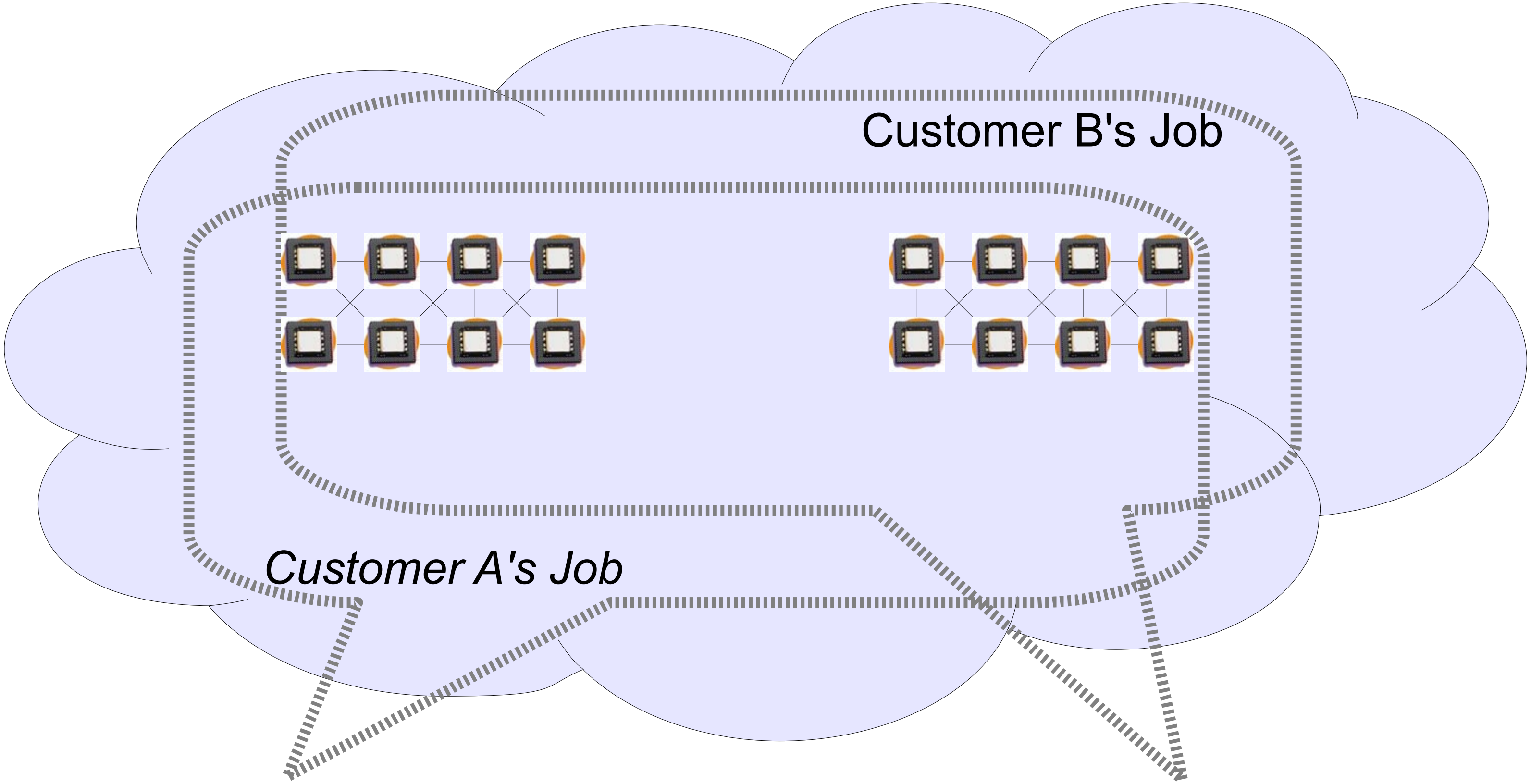
Allow modulation, **eliminate reference clocks**



Our Approach

Allow modulation, **eliminate reference clocks**

- *Dynamic statistical multiplexing allowed*



Timing Information Flow Control [HotCloud '12]

Adapt IFC to label & control timing channels

Key idea: separate labeling of *state* and *events*

- **State labels** attached to *explicit program state*
  - Represent ownership of information in the *content* of a variable, message, process, etc.
- **Time Labels** attached to *event channels*
  - Represent ownership of information affecting *time* or *rate* events occur in a program

Relies on **enforceable deterministic execution**

Timing Control in Elastic Clouds

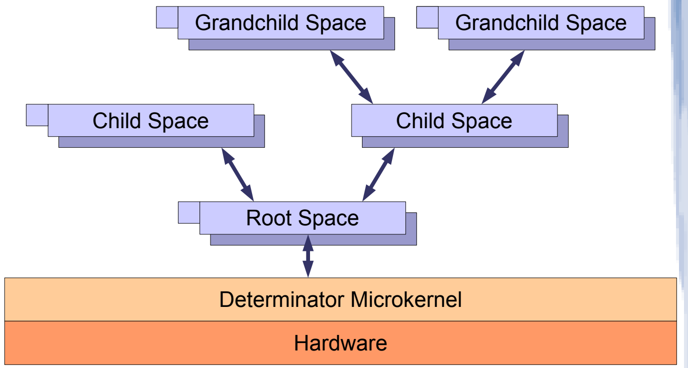
Need two key facilities:

- *System-enforced deterministic execution*
  - OS/VMM ensures that a job's outputs depend *only* on job's explicit inputs
- *Pacing queues*
  - Input jobs/messages at any rate
  - Output jobs/messages on a *fixed schedule*

Determinator

A Determinism-Enforcing Microkernel/Hypervisor

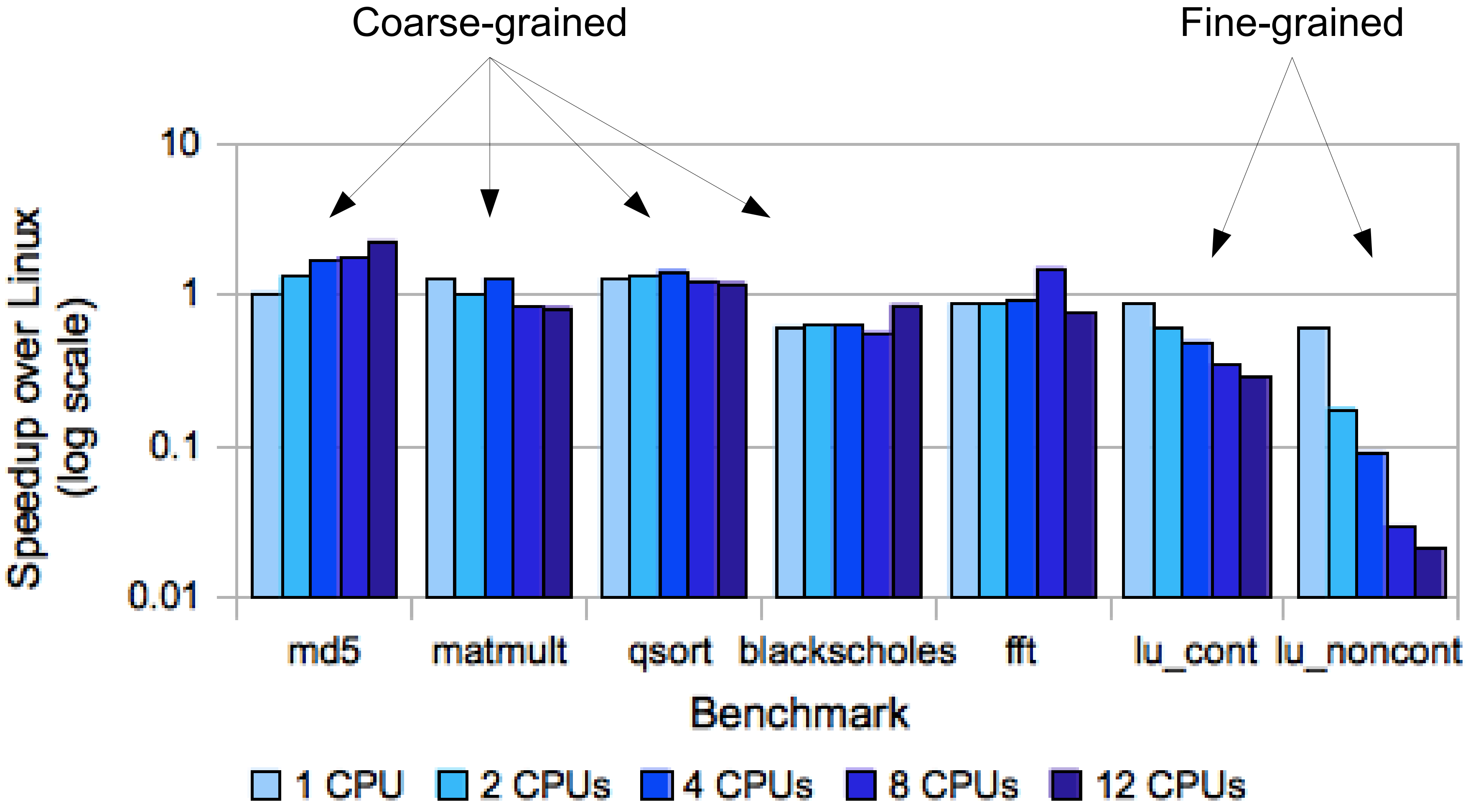
- ***"Efficient System-Enforced Deterministic Parallelism"*** (Best Paper Award, OSDI 2010)



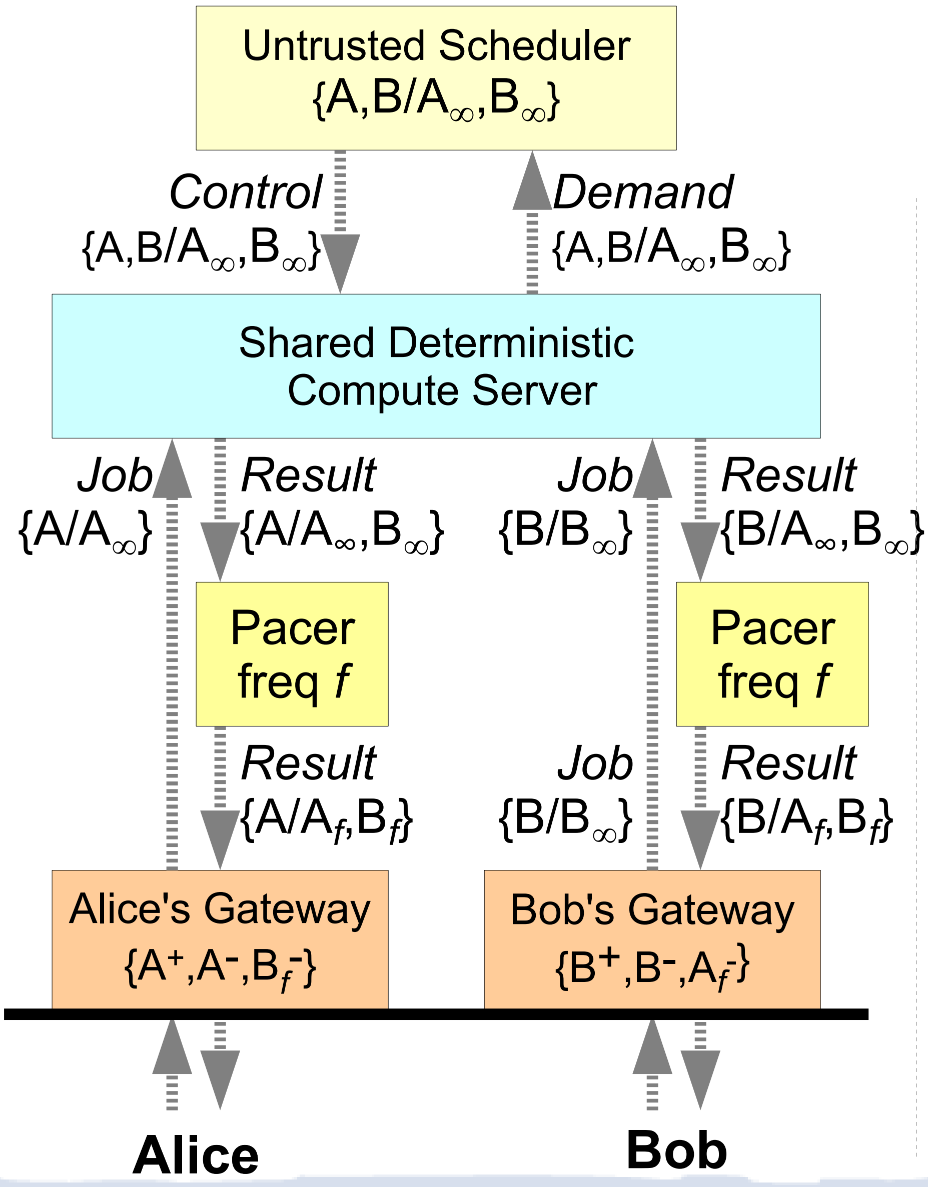
Enforces determinism on **parallel** applications

- Even if user code behaves adversarially
- Not provided by user-level approaches (DMP, CoreDet, Grace, Dthreads, etc.)

Determinator versus Linux



Elastic Cloud Scenario



Untrusted Scheduler
$\{A,B/A_\infty,B_\infty\}$

Control
$\{A,B/A_\infty,B_\infty\}$

Demand
$\{A,B/A_\infty,B_\infty\}$

Shared Deterministic
Compute Server

Job
$\{A/A_\infty\}$

Result
$\{A/A_\infty,B_\infty\}$

Job
$\{B/B_\infty\}$

Result
$\{B/A_\infty,B_\infty\}$

Pacer
freq *f*

Pacer
freq *f*

Result
$\{A/A_f,B_f\}$

Job
$\{B/B_\infty\}$

Result
$\{B/A_f,B_f\}$

Alice's Gateway
$\{A^+,A^-,B_f^-\}$

Bob's Gateway
$\{B^+,B^-,A_f^-\}$

Alice

Bob

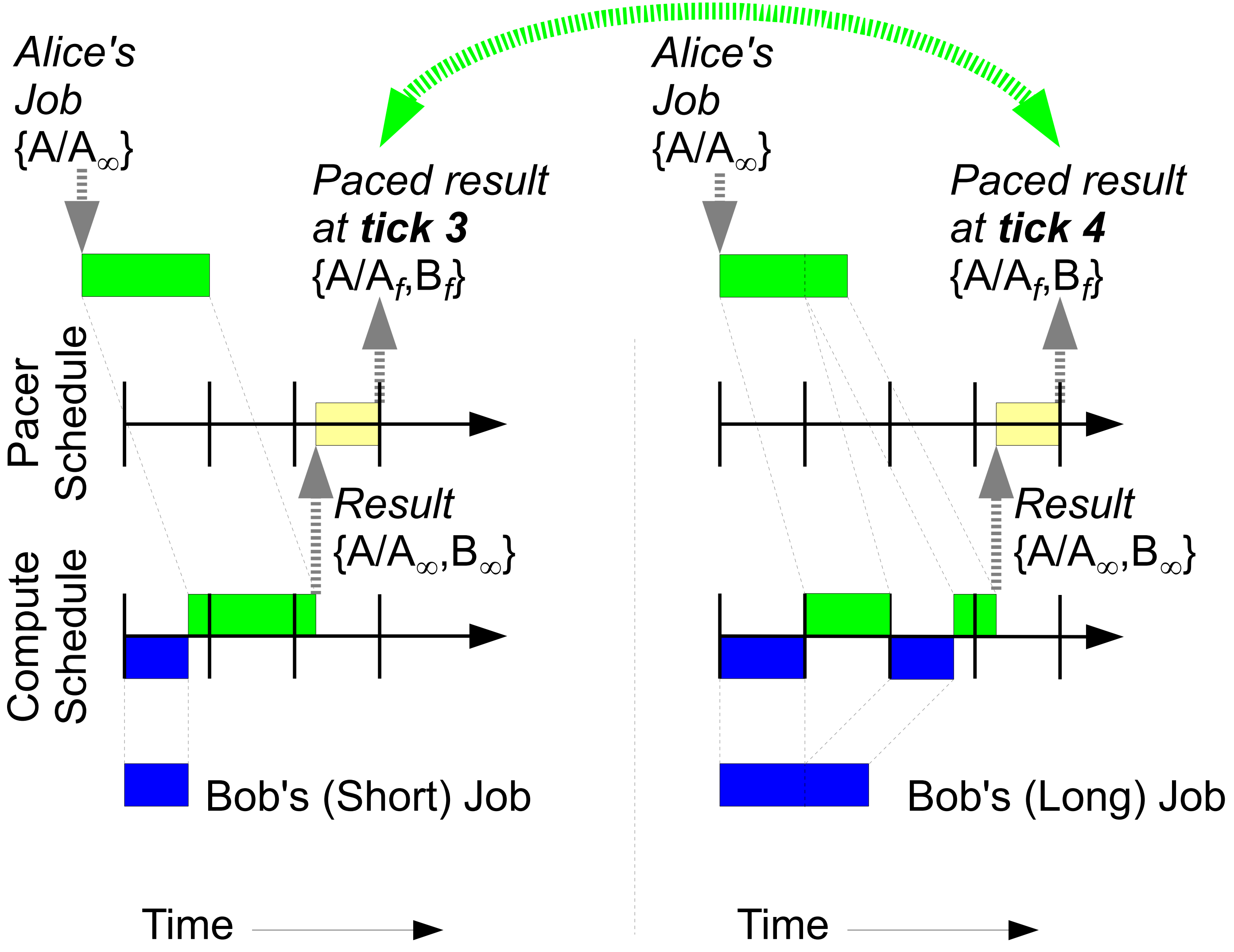
Jobs: In Anytime, Out on a Schedule

For each customer (e.g., Alice):

- Deterministic execution ensures job output *bits* depend only on job input *bits*: $O_j = f(I_j)$
- Job outputs produced *in same order* as inputs
- At each "clock tick", paced queue releases either ***next job output*** or says ***not ready yet***
  - The ***single bit of information*** per clock tick that might leak other users' information

Also supports **predictive mitigation** [CCS '11]

Informal “Schedule Analysis”



(b) Schedule: Bob's job short

(b) Schedule: Bob's job long

Key Challenges/Questions

- Formalize full TIFC model
  - Potentially applicable at systems or PL levels
  - Integrate Myers' “predictive mitigation” ideas
- Complete TIFC-enforcing prototype
  - Ongoing, based on Determinator [OSDI '10]
- Explore flexibility, applicability of model
  - Can model support interactive applications?
  - Can model support transactional apps?

Conclusion

First approach to timing channels control that:

  - Works with *unmodified* hardware and software
  - Works with *general* computing algorithms
  - Supports stat-multiplexed elastic computing

More info: http://dedis.cs.yale.edu/2010/det/

- “Determinating Timing Channels” [CCSW '10]
- “Plugging Side-Channel Leaks” [HotCloud '12]
- “Efficient System-Enforced Det...” [OSDI '12]