HARVARD
School of Engineering
and Applied Sciences

Language-based Information Security

CS252r Spring 2012

This course

- Survey of key concepts and hot topics in **language-based information security**
  - The use of programming language abstractions and techniques to reason about and enforce information security guarantees
- Aim: understand, and contribute to, the research boundary of the field
- Prereq: CS 152 or equivalent

Class meetings

- Meet twice weekly
- Combination of lectures and paper presentation/discussion
  - Lectures for background material/information not covered well by one or two papers
    - Will often include additional/relevant/recommended reading
  - Papers for recent research, case studies, exemplary approaches, …
  - Expect to present once (maybe twice) during semester
- Volunteers needed to present
  - Thursday Feb 9 onwards
  - Look at the schedule, and email me if you would like to present one of the papers.

Assessment

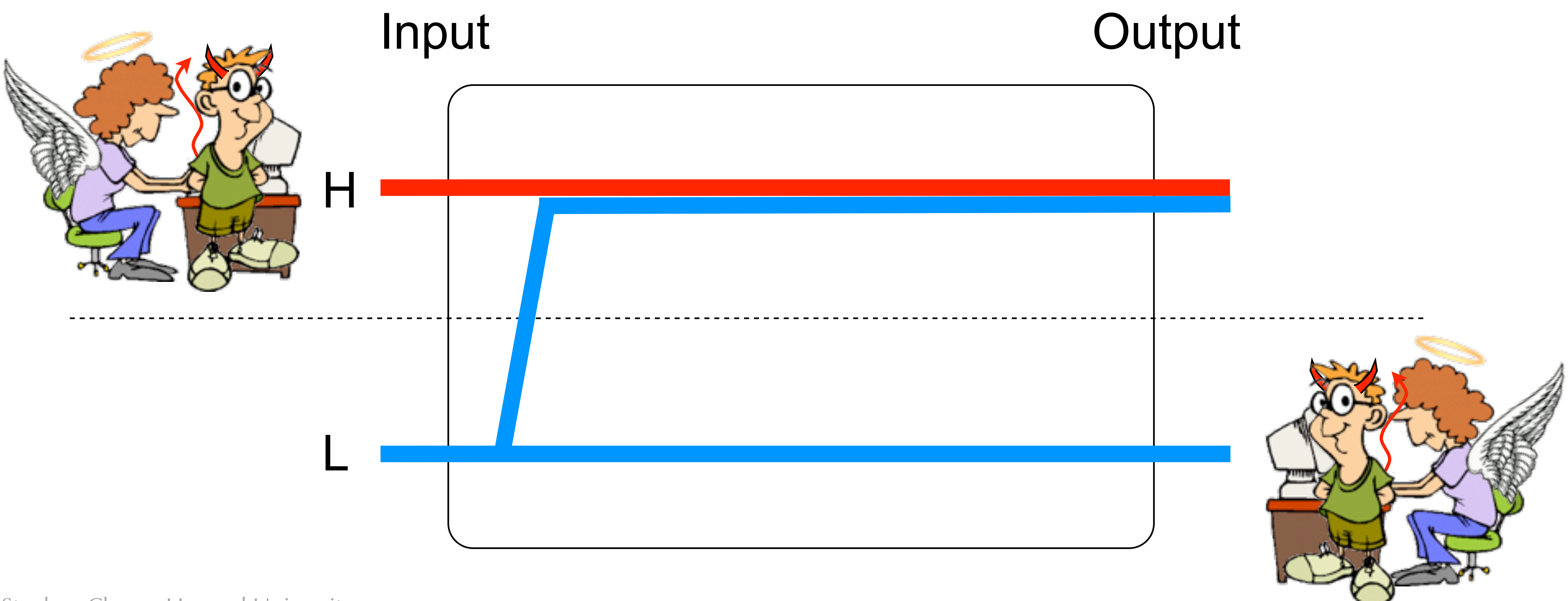
- Class participation
  - Presentation/discussion
- Project
  - Dig deep into one or more aspects of material covered in class
  - Encourage to work in teams of 2–4 people
  - From week 3 onwards, I will meet weekly with each team
  - Project proposal due Tuesday Feb 21 (week 5)
  - Project presentations April 17, 19, 24
  - Final report Thursday May 3
- Auditors welcome
  - We can discuss what level of participation is involved

Schedule

- See website
- Subject to change. Feel free to suggest papers/topics

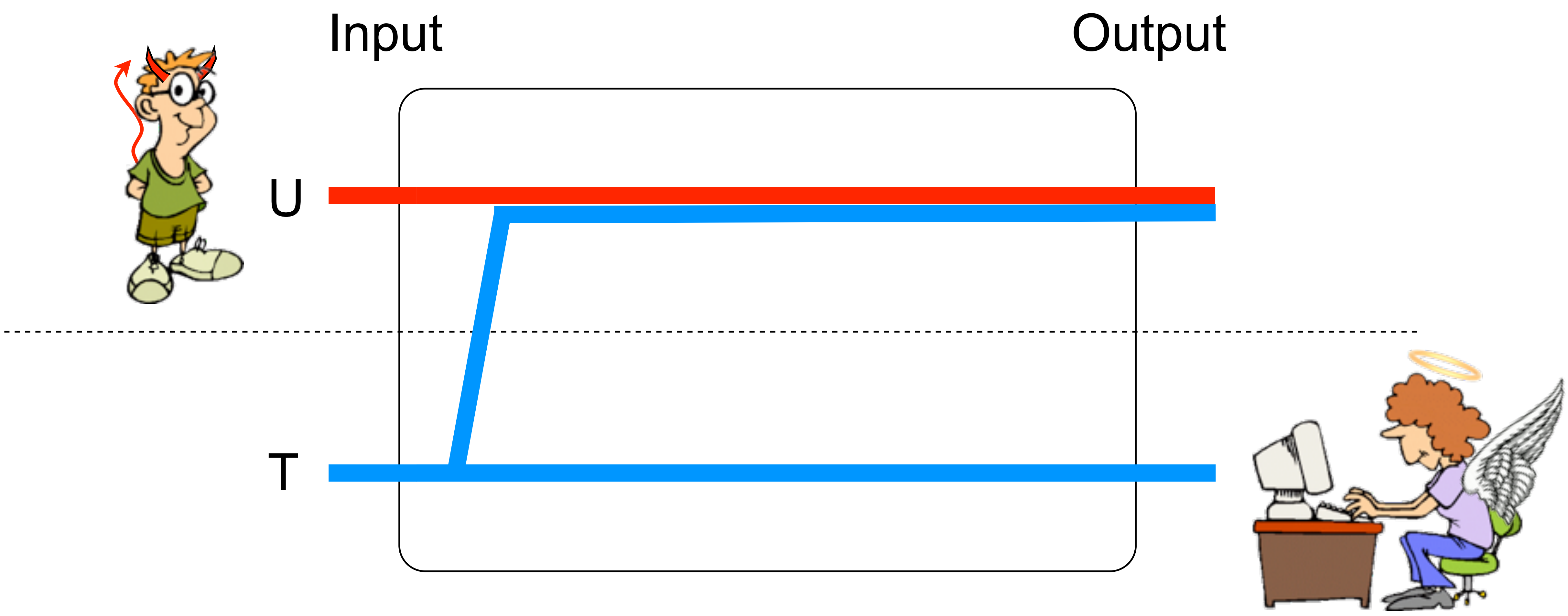
Information flow

- Information flows through systems
- We want to both understand how this information flows, and possibly restrict it



Information flow

- Information flows through systems
- We want to both understand how this information flows, and possibly restrict it



Uses of information-flow control

- Information flow is really about **dependency**
  - How does the output of a system depend on the input?
  - How does the input of a system influence the output?
- Very general concept.
- Many possible uses:
  - Stop confidential information from being released inappropriately
  - Stop untrusted information from being used inappropriately
    - SQL/command injection attacks, cross-site scripting attacks
    - Integer vulnerabilities
  - Provenance
    - Record the history of information/computation
    - Enables auditing, recomputation, querying, ...

So, what's left to learn?

- How does information flow in a system?
  - And why do we use language-level abstractions?
- Information-flow based semantic definitions
  - What does it mean to be "secure"?
  - What does it mean for information to "flow" or for an output to depend on an input?
- How do we enforce information-flow based notions of security?

What is information?

- For our purposes, bits in context
- E.g., consider following program
  - ```
    x = input_from_user();
    y = x + 1;
    z = y * -1;
    ```
  - Suppose we know that the value in program variable z at the end of the program is integer -43.
  - This allows us to work out the input supplied by user
  - The value -43, without context, doesn't tell us much at all...
    ```

How does information flow?

- **Explicit flow**
  - Flow through copying data or computation on data
  - e.g., y = x
    - Knowing the bits in y at that program point tells us exactly the bits in x at that program point
  - e.g., y = x+1
    - Ditto
  - e.g., y = x mod 8
    - Knowing the bits in y at that program point tells us something about the bits in x at that program point (the last 3 bits)
  - Non example: y = x * 0

How does information flow?

- **Implicit flow** (control flow channels)
  - e.g.,

    ```
    if (x == true)
        y = true
    else
        y = false
    ```

    - At end of this statement, value in y is the same as value in x at beginning of statement
  - e.g., y = 0; while (x > 0) { y++; x--; }
    - Ditto

How does information flow?

- Termination channels
  - Whether the program (or part of a program) terminates may reveal information
  - e.g., while (x > 0) { skip }; output “Hello!”
    - If “Hello!” is output, we know that $x \leq 0$
- Timing channels
  - How long a program (or part of a program) takes to execute may reveal information
  - e.g., output “start”; while (x > 0) { x--; }; output “stop”
    - How long between outputs may reveal information about initial value of x
- Other covert channels
  - Often not at a PL level of abstraction
  - Power consumption, processor noise, temperature, ...

Why use language-level abstractions?

- Information-flow control at programming language level of abstraction
  - Fine-grained
  - Can soundly control implicit flows
  - Clean semantics
  - Language techniques
- Coarser levels of abstraction cannot distinguish reliably distinguish sensitive bits from non-sensitive bits
  - Language-level approaches provide finer-grained, human meaningful "contexts"

Semantic definitions of security

- **Noninterference**
  - Intuitively, confidential inputs do not influence (or “interfere with”) public outputs
  - Integrity version: untrusted inputs do not influence trusted outputs
  - Availability version: outputs that should be highly available do not depend on inputs that are not highly available
- Some problems and issues with noninterference
  - We will consider these in later classes…

Formally defining noninterference

- Goguen and Messeguer 1982 define noninterference in terms of sets of users. Users U are *noninterfering* with users V if the commands issued by U does not change the observations made by V.

Formally defining noninterference

- More common modern formulation is using **pairs of executions**
  - Definition: Program c is **noninterfering** if for all states $\sigma_0, \sigma_1, \sigma'_0, \sigma'_1$:

    if

    $$\sigma_0 \approx_L \sigma_1 \text{ and } [\![c]\!]\sigma_0 = \sigma'_0 \text{ and } [\![c]\!]\sigma_1 = \sigma'_1$$

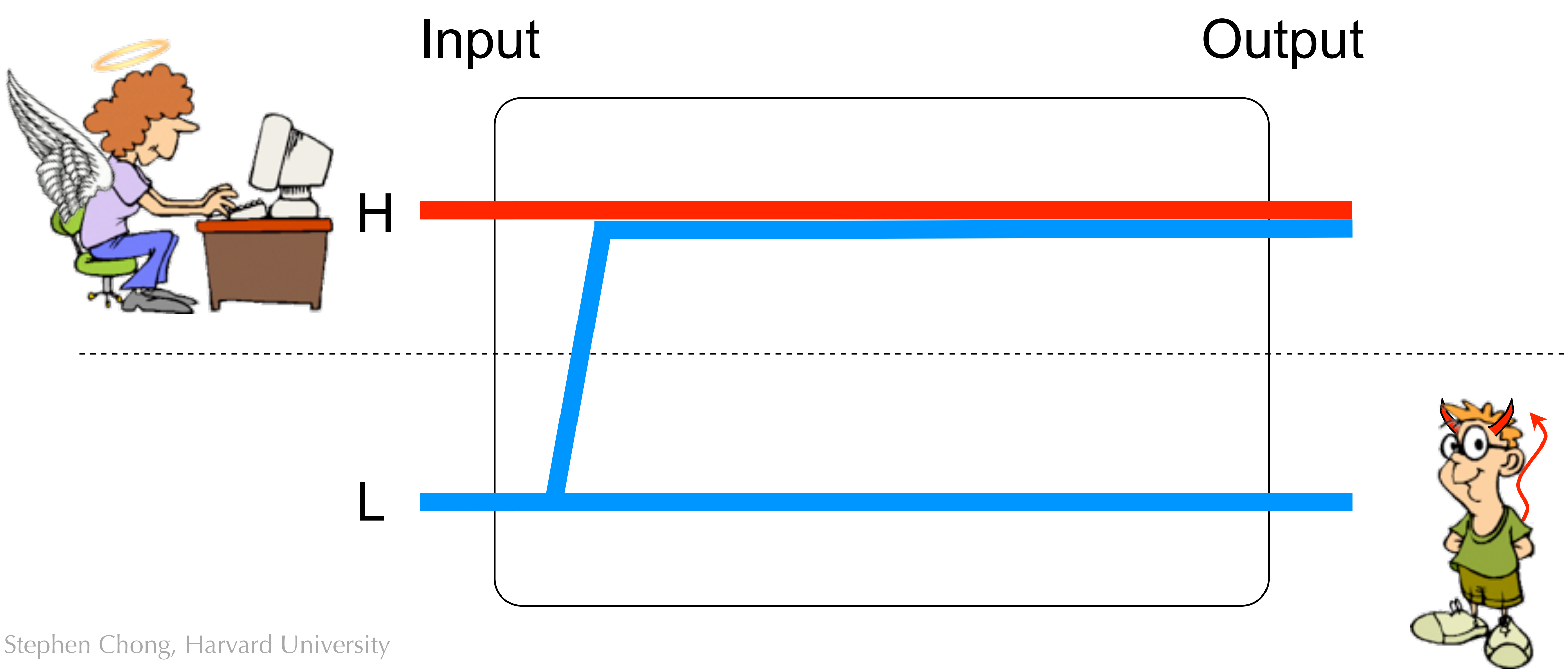
    then

    $$\sigma'_0 \approx_L \sigma'_1$$

  - Here $\approx_L$ is **observational equivalence**
    - $\sigma \approx_L \sigma'$ iff $\forall x \in \text{ObsVars}.\ \sigma(x) = \sigma'(x)$

Observational model

- An explicit **observational model** helps us understand the semantic security condition
  - What can the attacker observe?
    - Memory locations? Outputs? Throughout execution? At beginning and end of execution? What about termination? What about timing information?



Interactive model

- Interactive programming models provide a more realistic model of observational behavior

$$
\begin{aligned}
e &::= v \mid x \mid e_1 \oplus e_2 \\
c &::= \mathsf{skip} \mid x := e \mid c_1 ; c_2 \\
&\quad \mid \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \mid \mathsf{while}\ e\ \mathsf{do}\ c \\
&\quad \mid \mathsf{input}\ x\ \mathsf{from}\ \ell \mid \mathsf{output}\ e\ \mathsf{to}\ \ell
\end{aligned}
$$

  - Channels are how the system interacts with its external environment
  - An attacker observes one or more channels

Interactive model semantics

$$\frac{m(e) = v}{\langle x := e, m, w\rangle \longrightarrow_{\epsilon} \langle \mathsf{skip}, m[x \mapsto v], w\rangle} \qquad \frac{\langle c_1, m, w\rangle \longrightarrow_{\alpha} \langle c_1', m', w'\rangle}{\langle c_1; c_2, m, w\rangle \longrightarrow_{\alpha} \langle c_1'; c_2, m', w'\rangle} \qquad \frac{}{\langle \mathsf{skip}; c, m, w\rangle \longrightarrow_{\epsilon} \langle c, m, w\rangle}$$

$$\frac{m(e) = i}{\langle \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, m, w\rangle \longrightarrow_{\epsilon} \langle c_i, m, w\rangle} \qquad \frac{}{\langle \mathsf{while}\ e\ \mathsf{do}\ c, m, w\rangle \longrightarrow_{\epsilon} \langle \mathsf{if}\ e\ \mathsf{then}\ (c; \mathsf{while}\ e\ \mathsf{do}\ c)\ \mathsf{else\ skip}, m, w\rangle}$$

$$\frac{w(\ell) = v : vs}{\langle \mathsf{input}\ x\ \mathsf{from}\ \ell, m, w\rangle \longrightarrow_{i(v,\ell)} \langle \mathsf{skip}, m[x \mapsto v], w[\ell \mapsto vs]\rangle} \qquad \frac{m(e) = v}{\langle \mathsf{output}\ e\ \mathsf{to}\ \ell, m, w\rangle \longrightarrow_{o(v,\ell)} \langle \mathsf{skip}, m, w\rangle}$$

- ω is **input strategy**: function from channels to input streams

$$\epsilon \restriction \ell = \epsilon$$

$$(\alpha \cdot t) \restriction \ell = \begin{cases} \alpha \cdot (t \restriction \ell) & \text{if } \alpha \in \mathbb{E}(\ell) \\ t \restriction \ell & \text{if } \alpha \notin \mathbb{E}(\ell). \end{cases}$$

$\langle c_0, m_0, w_0\rangle \Downarrow_\ell t$ if there are k configurations $\langle c_i, m_i, w_i\rangle$ for $i \in 0..k$ such that

$$\langle c_{i-1}, m_{i-1}, w_{i-1}\rangle \longrightarrow_{\alpha_i} \langle c_i, m_i, w_i\rangle$$

for all $i \in 1..k$, and $t = (\alpha_1 \cdot \ldots \cdot \alpha_k) \restriction \ell$.

Knowledge-based definitions

- The "pairs of execution" definitions are somewhat unintuitive
  - Trying to capture the idea that an attacker cannot distinguish executions that differ only on secret values, and thus cannot learn the secret.
- Why not express this more directly?
- **Knowledge-based definitions** explicitly define attacker knowledge, and define security in terms of attacker knowledge.

Knowledge

- **Attacker knowledge** $k(c, \ell, t)$ is the set of input strategies that could have resulted in trace t being emitted on channel ℓ

$$k(c, \ell, t) = \{w \mid \langle c, m_{init}, w \rangle \Downarrow_\ell t\}$$

- $k(c, \ell, t)$ is the set of input strategies that an observer of channel ℓ believes are possible after observing trace t
  - Smaller set = more precise knowledge
  -

Knowledge-based security

- Define $\omega \approx_{\sqsubseteq \ell} \omega'$ if $\forall \ell' \sqsubseteq \ell .\ \omega(\ell') = \omega(\ell)$
  - i.e., ω and ω' agree on all inputs $\ell' \sqsubseteq \ell$

- Program c is satisfies noninterference for channel ℓ

  if

  for all input strategies ω,

  if $\langle c, m_{init}, \omega \rangle \Downarrow t$

  then $k(c, \ell, t) \supseteq \{\omega' \mid \omega \approx_{\sqsubseteq \ell} \omega'\}$

Next class

- Enforcing noninterference
  - Static, dynamic, and hybrid techniques
- Lattice based policies