Outline

Formal Methods

Agents

Security Policies

Secure Mobile Code Infrastructure and Types

Protecting Machines from Agents
and Agents from Machines

Patrick Lincoln

SRI International
333 Ravenswood Avenue
Menlo Park CA 94025

http://www.csl.sri.com

Outline

Formal Methods

Agents

Security Policies

What Next For Formal Methods?

More Automation

Less Intrusiveness (Invisible?)

More Expressive Types

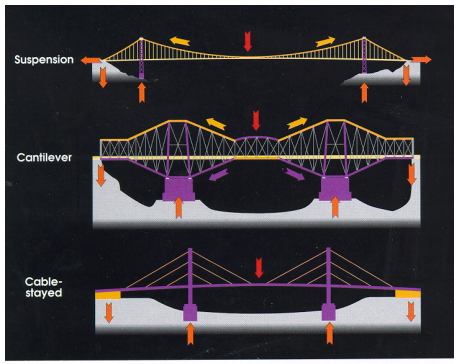
Application to Mobile Code

Application to Other Domains
Symbolic Systems Biology

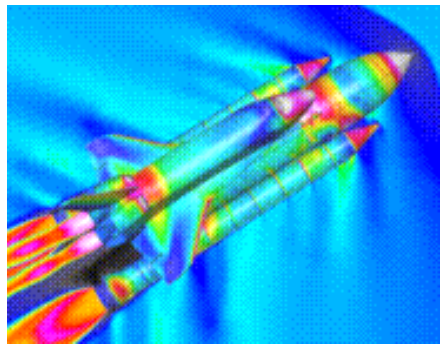
Test at Design, Integration, and/or Run Time

In some fields, mechanized design analysis works

Abstract mathematical model Subject to calculation



Finite-element model



Navier-Stokes model

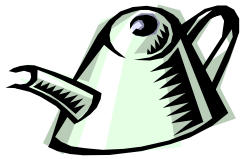
?

Complicated system

Digital Hardware and Software Artifacts

Present Costs and Benefits

Formal verification is expensive, testing incomplete



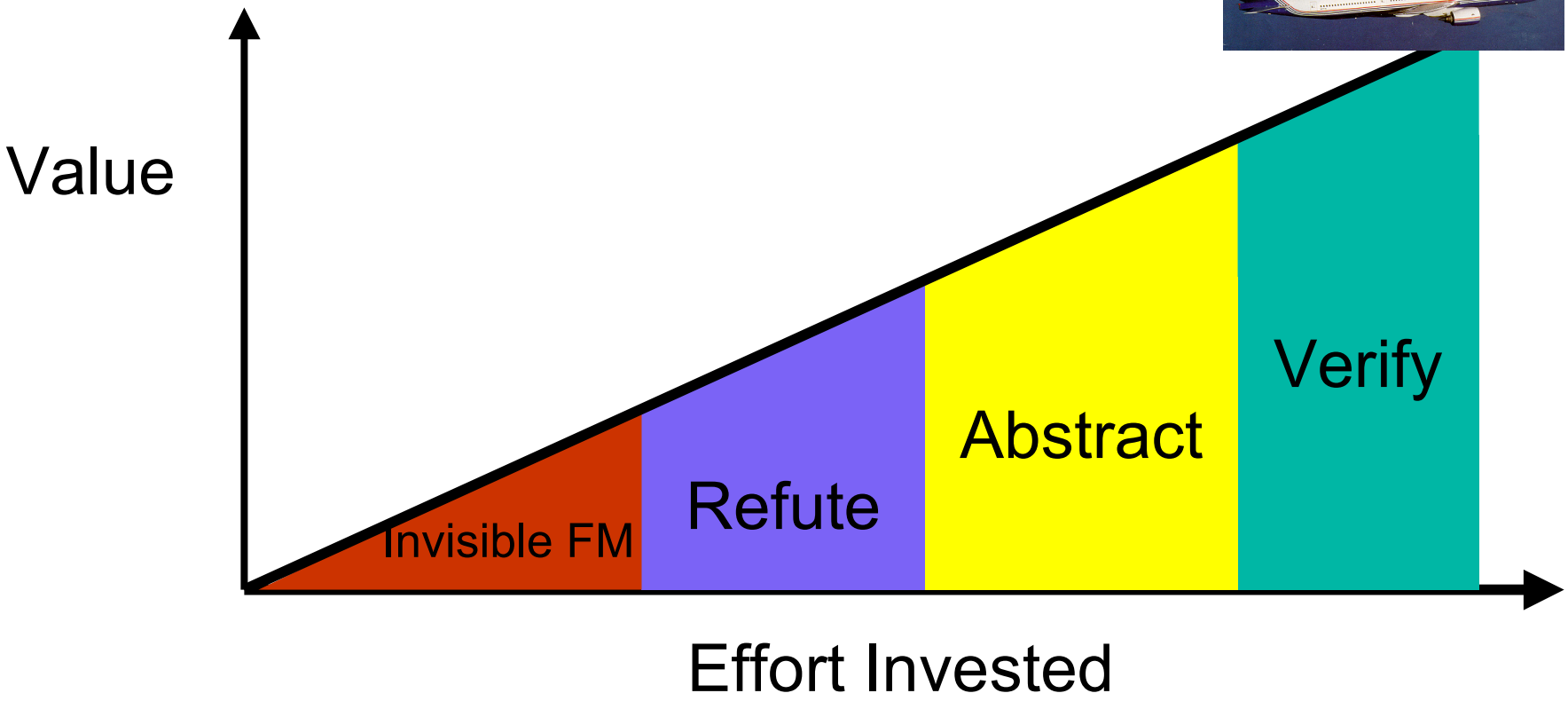
Value

Testing

Verification

Effort Invested

Dialing up and down accuracy of analysis

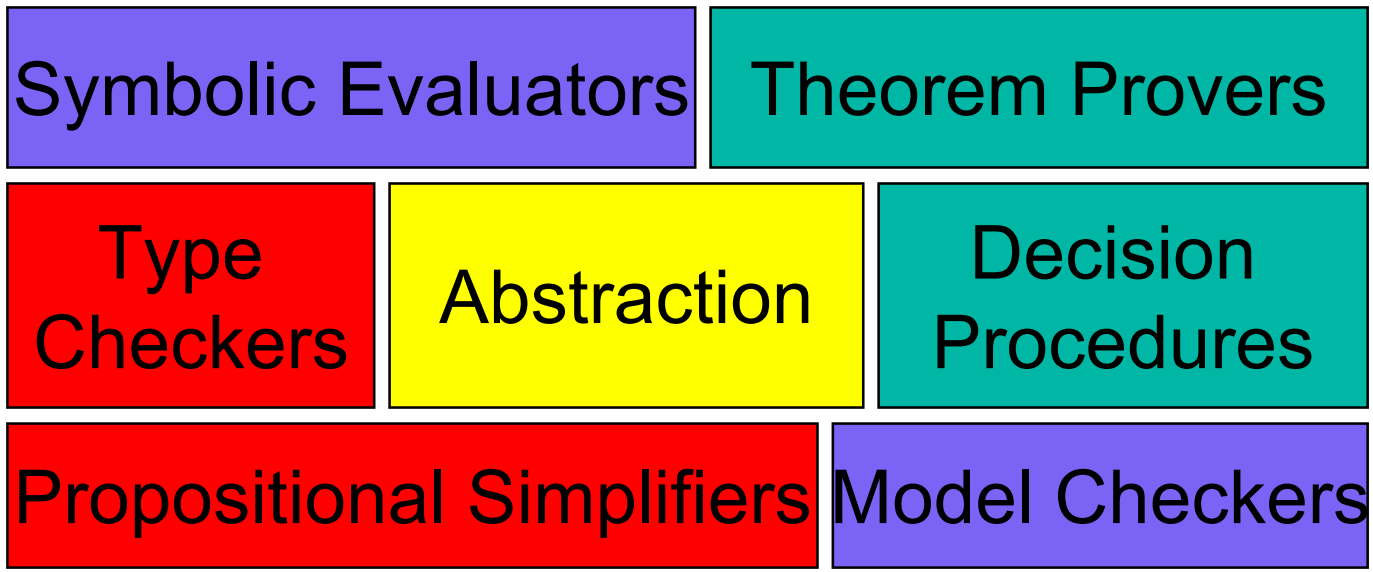


Components

Integrated Canonizer and Solver (ICS)

PVS Type Checker

Symbolic Analysis Laboratory



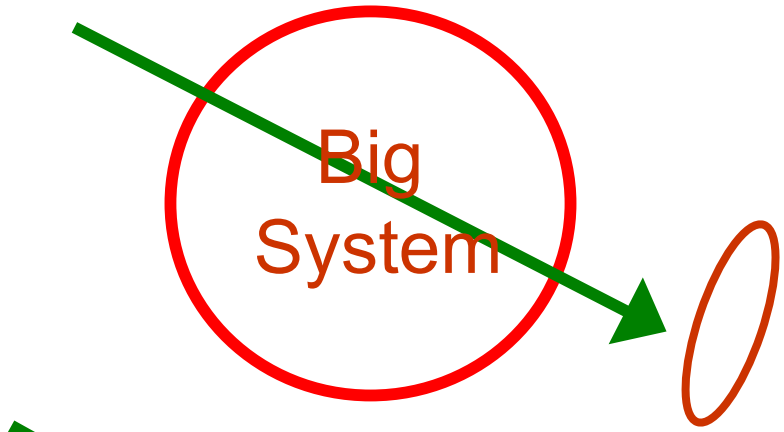
Key Lever: Property-Sensitive Abstraction

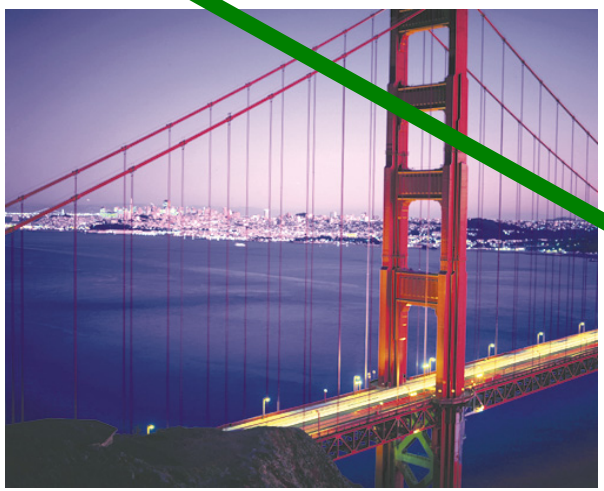
Property-sensitive abstraction

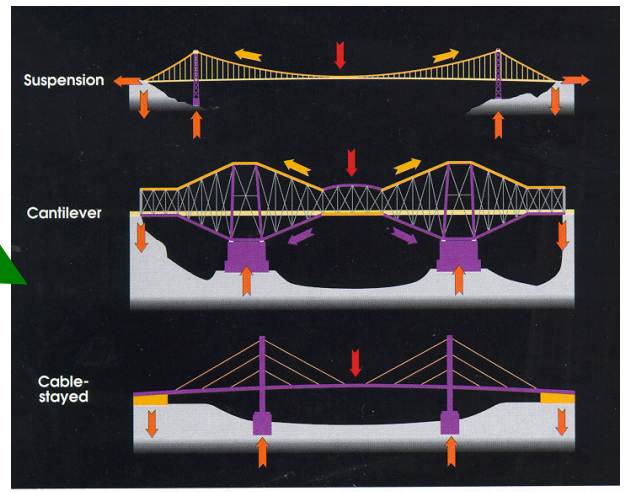
Tractable analysis of the simplified system

Model checking

Integrated decision procedures







Property-Sensitive Abstractions

Ways to simplify

Dimensions of projection

- Scaling
  - From infinite to finite to small to trivial
  - Discretizing continuous (hybrid) systems
- Slicing
  - Ignore most of a system
- Separation of concerns
  - Correctness
  - Termination
  - Real time
  - Fault tolerance
  - Security

Abstraction Not Enough: Need Concretization

Abstractors and concretizers enable iteration

Murphi

Maude

XML

CAPSL

CIL

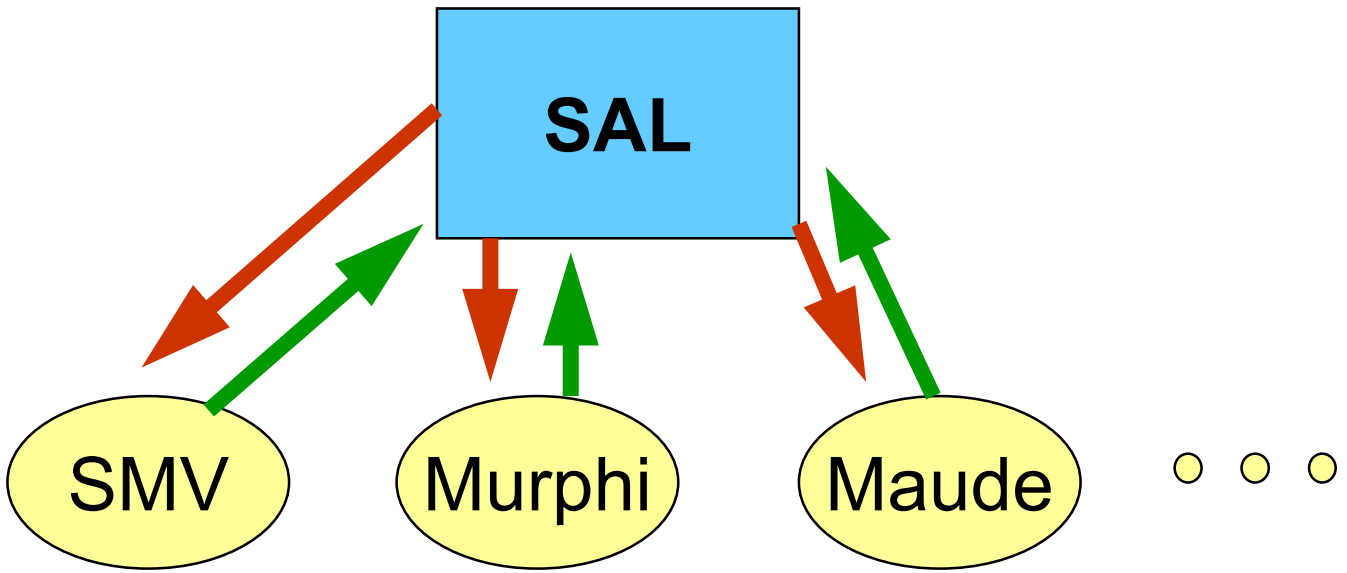
HOL

NRL Protocol Analyze

Chaff

...

PVS Symbolic Analysis Laboratory: abstractors and concretizers



PVS Applications

Post-hoc analysis of certain aspects of big systems

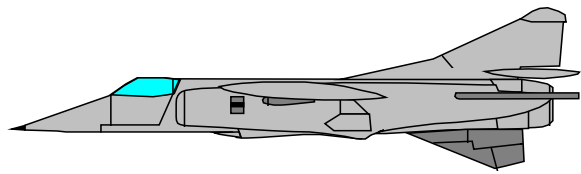
- Pentium FDIV
- NASA RCP
- Space shuttle jet select
- Allied Signal's Z: found critical bug

Analyzing certain properties of abstract system designs

- OMH
- ZA, OMHA: Signed versions
- Spacecraft control FTTP: Draper's Agreement

Diagnosis agreement, and scheduling algorithms for fault-tolerant architectures

- Swim-by-wire: HD: Allied-Signal's Diagnosis
- 777 Safebus: fly-by-wire
- TTA: Kopetz' drive-by-wire
- Swing-wing control for F14



Application to Mobile Code

If we don't even care enough about an agent to even give it it's own hardware, how can we expect it to be secure & reliable?

So, what can we do for agents?

What is an Agent?
A program and some state

Examples

Voyager, Aglets, Odyssey

Robots, Softbots

FireFly

ActiveX, Microsoft Agent, Julia

ModSAf

KQML, FIPA

Klaim, Ambients

Ajanta

OAA, Cougaar

Mobile Maude

Mobile Agents

Programs that move among computer hosts

Autonomous Agents

Based on planning technologies

Learning Agents

User preferences, collaborative filtering,...

Animated Interface Agents

Avatars, chatbots, ...

Simulation-based Entities

Cooperative Agents

Collaboration among distributed heterogeneous components

Why?

Personalization

Reliability

Always-on

Always connected

Always controllable

Where are Agents?

Server

Database, corporate knowledge bases

Desktop

Personal shopping agents

The Wired Network

The edge computing fabric

Palmtop

Wearable agent technologies

Wireless Network

Personal-area network

Consumer Electronics

Web-enabled X $\Rightarrow$ Agent-enabled X

Key Properties

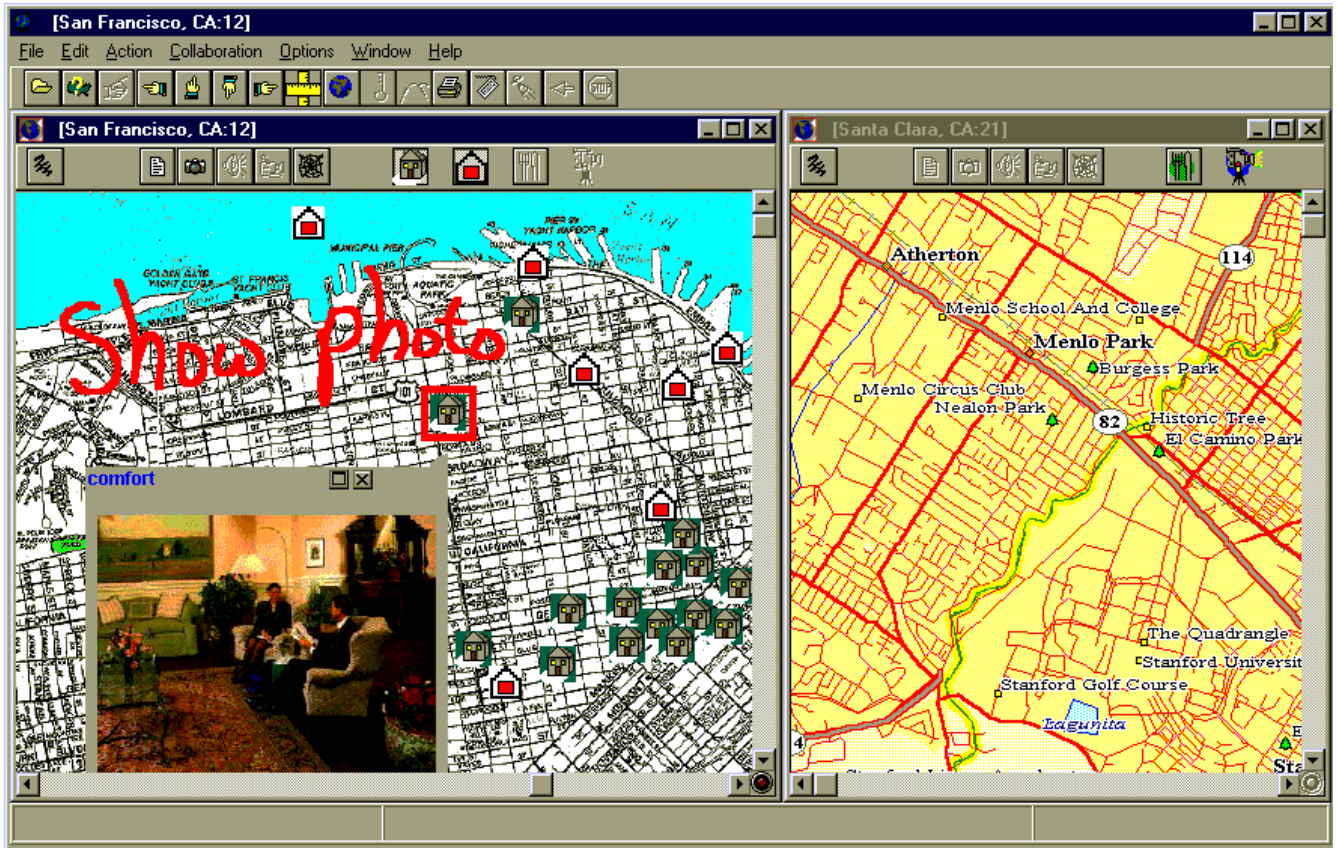
Natural interface to distributed (web) data

Synergistic combination of handwriting, drawing, speech, direct manipulation

Parallel cooperation and competition among many agents

Human & Agent collaboration

Multimodal Maps Application



Automated Office Application

Main Points

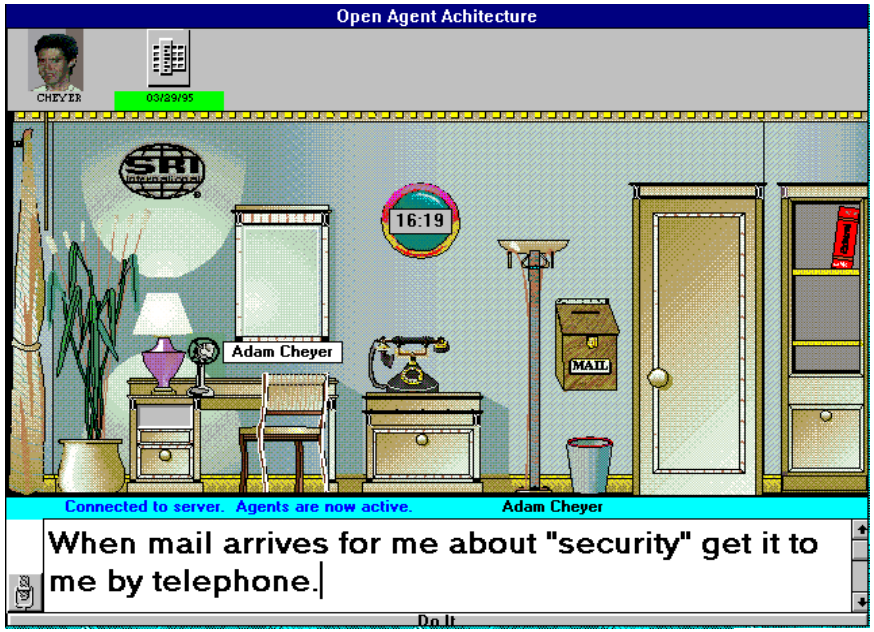
Mobile access to distributed services

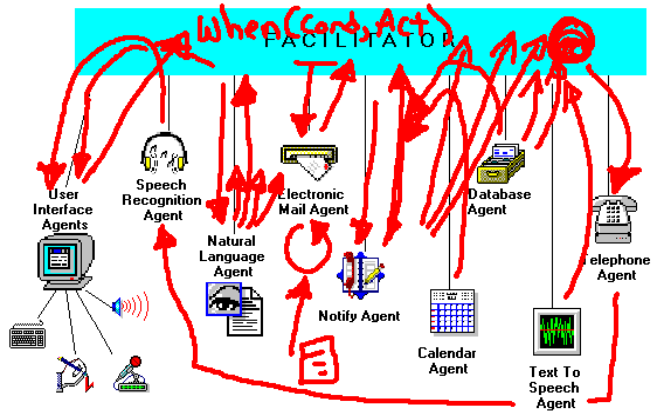
Legacy applications interacting with AI technologies

High-level tasking of agents through NL and speech

Flexible interactions among components

Delegated Triggers





That's All Great, but What About Security?

Threats

Malicious Attackers

Script Kiddies

HW/SW Faults

Accidental Misconfiguration

Insider Threats

Mobile Agents

New threats: more capable virus

Autonomous Coordinated Agents

Agents colluding to delude the user

Learning Agents

Highly adaptive adversary in your network

Animated Interface Agents

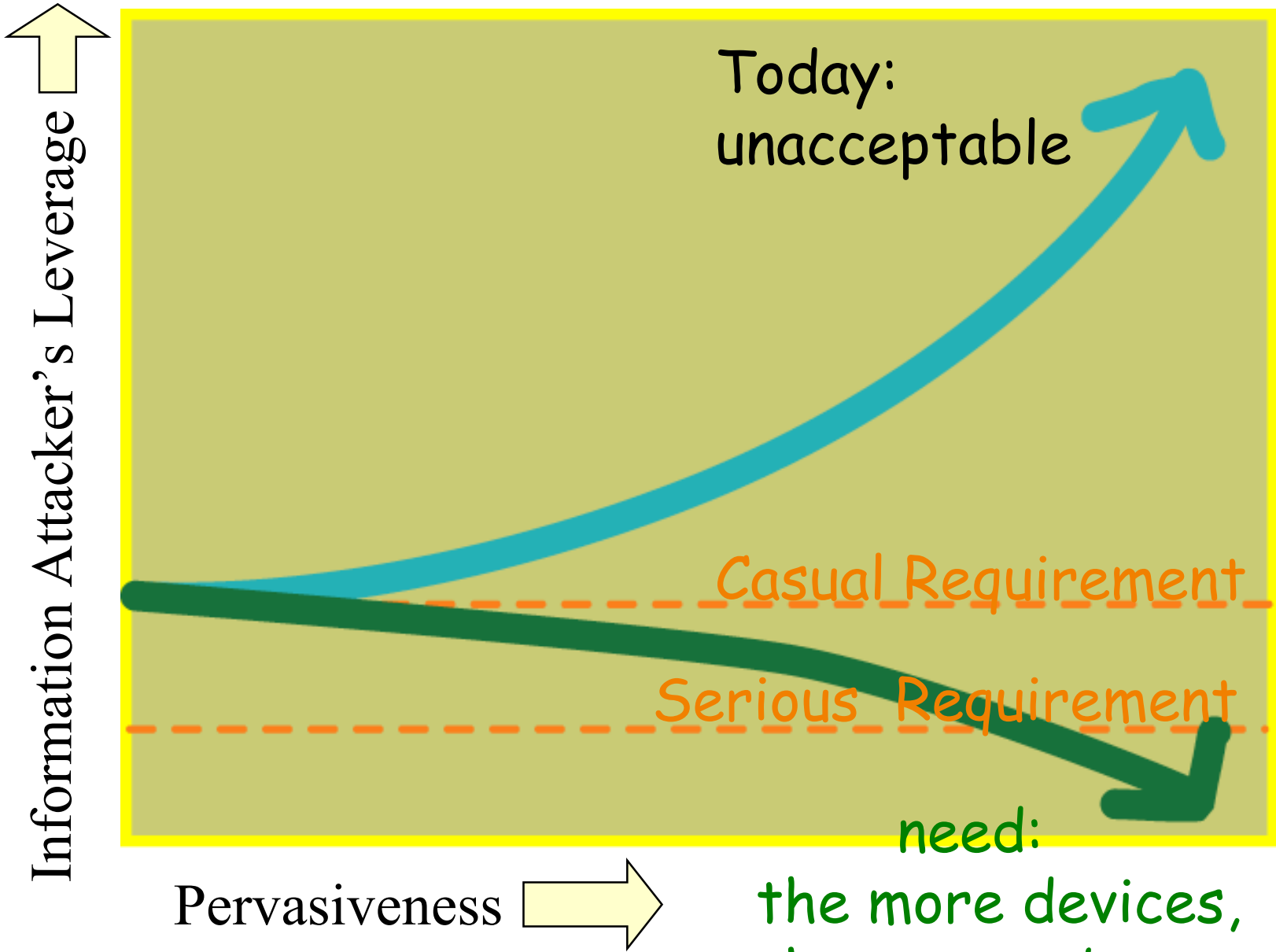
Someone else's code between you and your computer

Emergent Behavior

New complex global behavior from simple individual agent behavior

Can be good, but could be very dangerous

Problem: insecurity increases with distribution



Protecting your machine

Virus

The more active, the more potential for destructive power

Java/Jini

Malicious code downloaded automatically

Subversion of active services

Computer break-ins, Intrusions

Fraud, insider abuse

Launch autonomous agent from the inside

ActiveX controls

Tricky to use securely (Authenticode, etc)

Threats

Malicious Hosts

ePickpockets

HW/SW Faults

Accidental Corruption

Insider Threats

And even if your machine is protected...

What about your agents?

If they go mobile to an untrusted host, are your agents safe?

Can a machine pick your agent's pocket?

Can valuable data, eCash, and authorizations be destroyed, modified, or stolen?

Can emergent behavior of societies of agents interfere with your mission?

Partial Solutions

PKI

Signed Communications

Encrypted Code

Monitoring of systems and insiders

Some tools being developed and deployed

Authentication

PKI, smart cards, tokens, etc.

Secrecy

Ubiquitous crypto

Integrity

Protect data, code, and state

Distributed Security

Threshold cryptography

Dynamic Monitoring

Processes and places

Secure Frameworks

Strong levels of security in the infrastructure

How to let Formal Methods help

Carry around formalized models of agent's behavior, and requirements on environment

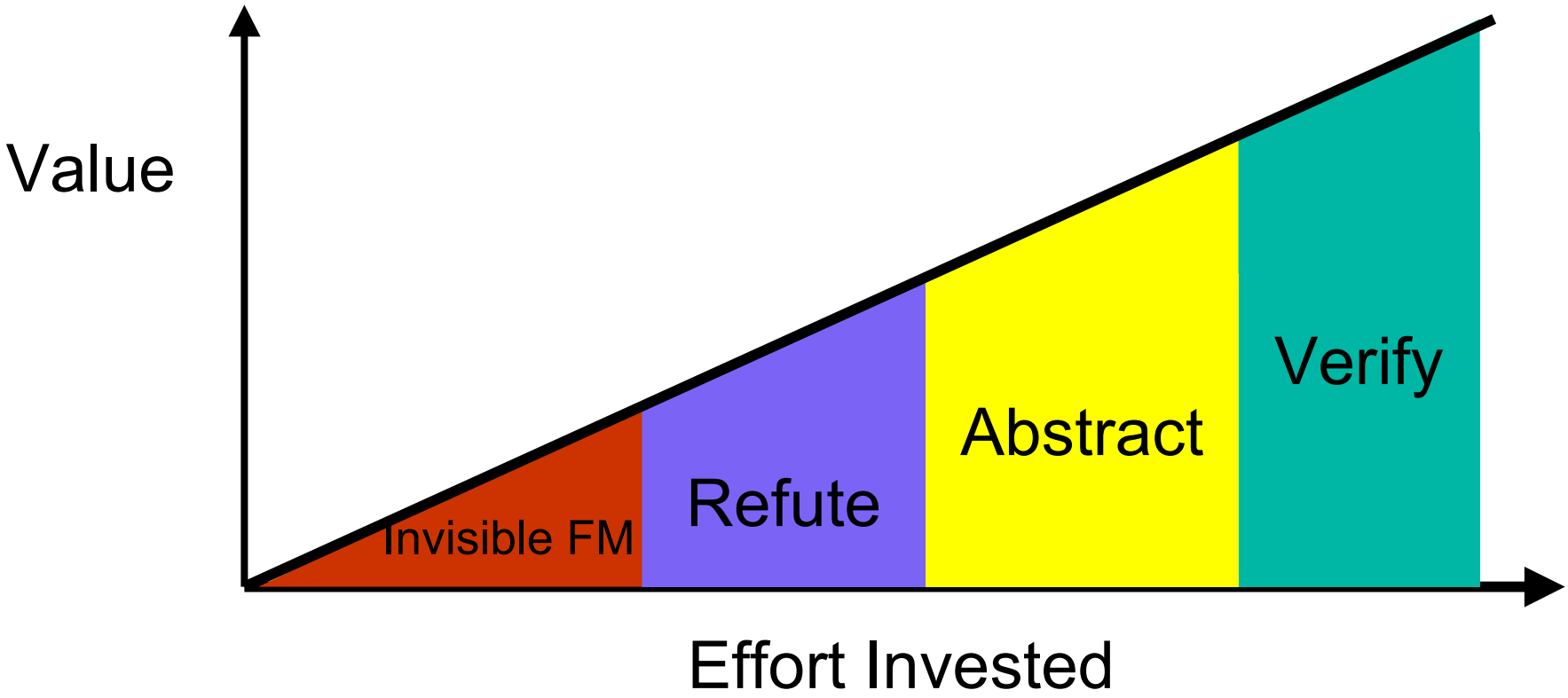
Revisit Definition of Agent:

- Code
- State
- Proofs (TAL, PCC, TILT, Ginseng)
- Formalized requirements

More Secure Frameworks

- Strong levels of security in the infrastructure
- Correctness by construction
- Extremely expressive type systems

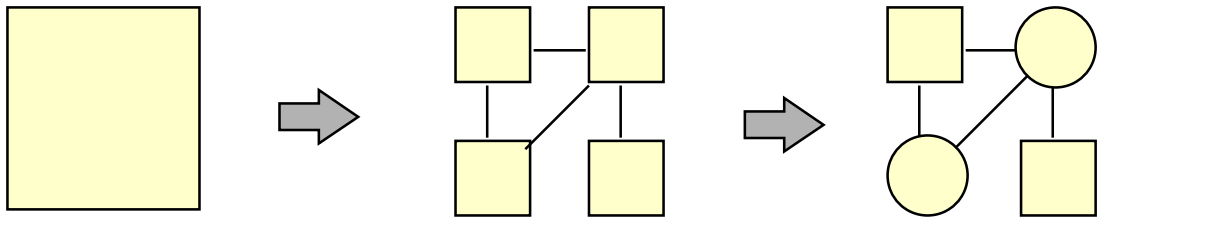
If we don't even care enough about an agent to give it it's own hardware, how can we expect it to be secure & reliable?



Some Problems

- Code certification
- Policy calculation
- Keeping secrets

Approaches to Building Agent-Based Applications



Monolithic Applications

Object-Oriented Applications

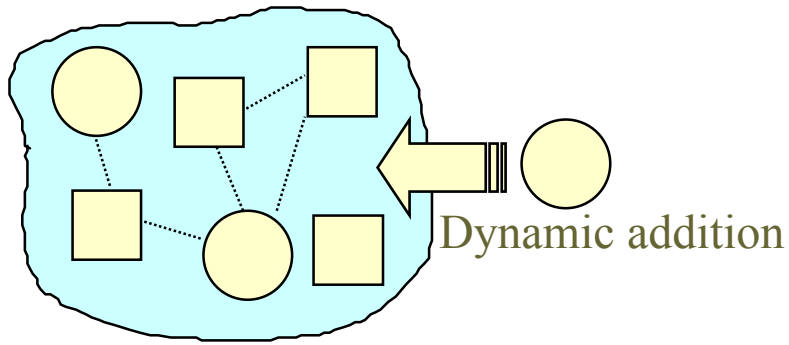
Distributed Object Applications

Objective

Virtual community of dynamic services

Adaptable to changing, evolving network resources

Flexible interactions among components



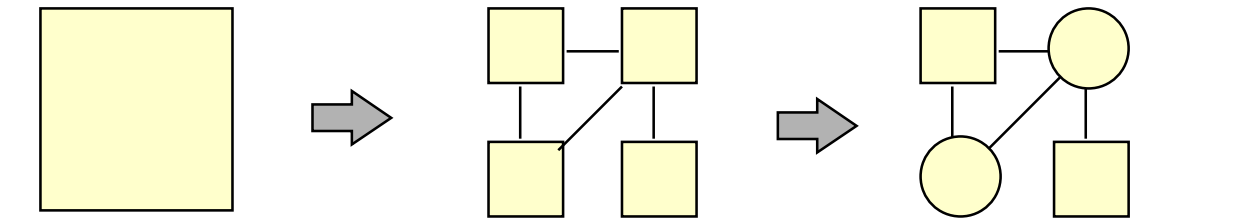
Agent-Based Applications

Approaches to Building Secure Agent-Based Applications

Objective

Trusted virtual community of dynamic services

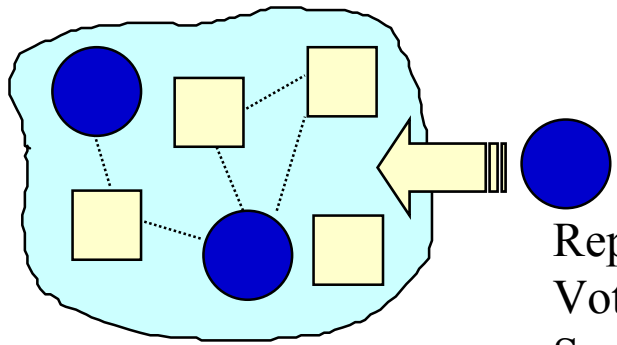
Adaptable to changing, evolving network resources with assurance



Monolithic Applications

Object-Oriented Applications

Distributed Object Applications



Secure Agent-Based Applications

Replication
Voting
Secret-Sharing
Program-Checking

A Secure Interagent Architecture

Agent Types

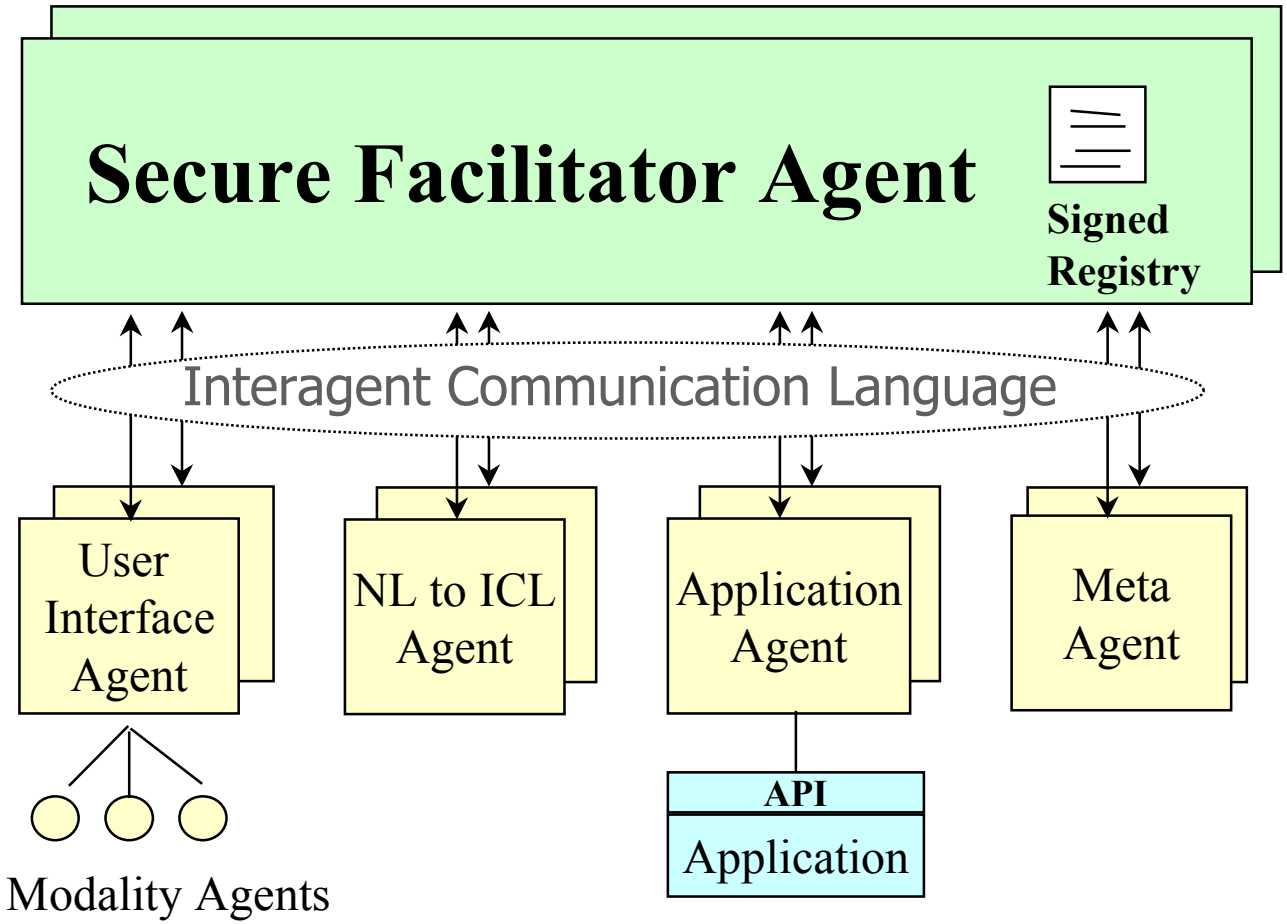
Some agents are replicated

Some secrets are shared

Some Agent communication is signed, encrypted

Facilitator Agents receive ICL requests and coordinate multiagent execution

Meta Agents help coordinate other agents, monitor emergent properties



Mobile Code Certification

- Assumptions:
  - The code producer is responsible to ensure compliance with requirements of the consumer.
  - The code consumer refuses to trust the assurances of the producer, but wishes to see for himself.
- An honest producer wants to comply with the consumer's requirements.

All methods to date certify safety with respect to a **type** system

Enforcement of abstraction

Tracking value ranges (eg, array bounds)

Problems

Generation of specification

Human specification

Automated additions

Generation of proof

Nearly invisible methods

Communication of proof

Proofs can be big

Bandwidth can be small

(probability / bandwidth tradeoff)

Underlying idea

- Hard to find a proof
  - Undecidable, NP-hard, …
- Easy to *check* a proof
  - Low-order polynomial time (n^2, n log n, ?)
- Proof-carrying code is
  - Program code
  - Assertion
  - Enough information to reconstruct proof

- Description-carrying code
  - with a checkable description

```
for (...) {
    ...
}

Assert f(x) = y
```

Not Checkable

```
for (...) {
Invariant ...
    ...
}
Assert f(x) = y
```

Checkable

- Hypotheses
  - Let L be some Turing-complete language
  - Let L' be decidable with all fctns total
    - (satisfying some expressiveness condition)
  - Let f be some total recursive function
- Conclusion
  - There exists a function g such that shortest program P' for g in L' is at least f(|P|) where P computes g in L

If assertions imply termination, pcc is arbitrarily large

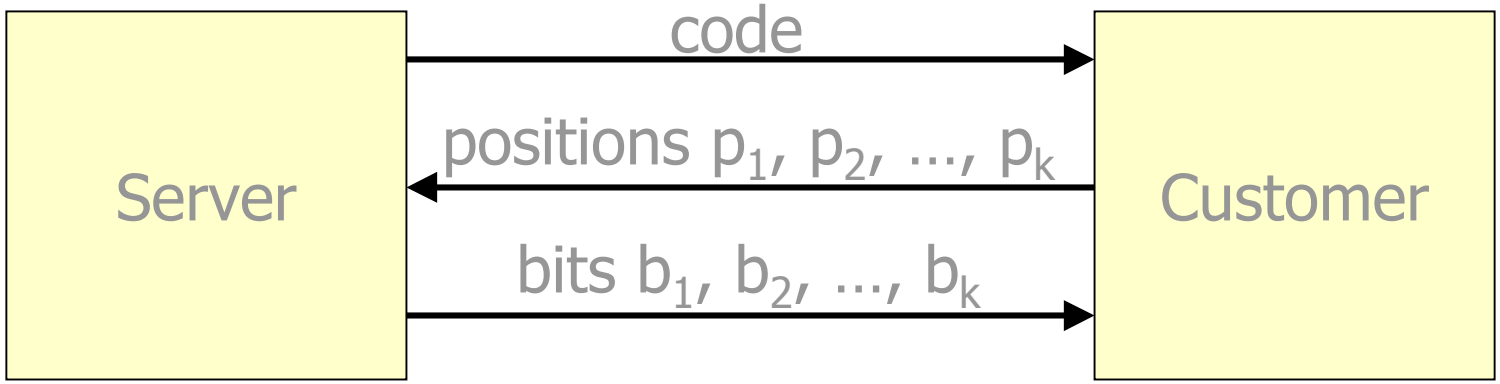
Probability/Bandwidth tradeoff

- OK, so proof might be big …
- Trade probability for bandwidth?
  - Some server has huge proof
  - Customer wants some guarantee
    - Willing to accept small probability error
    - If not necessary to receive and read entire proof

- Every $L \in NP$ has prob checkable proofs
- Specifically,
  - for any language $L \in NP$ and any string x,
  - there is a candidate proof (another string) π_x with length polynomial in |x|,
  - π_x can be used to prob check whether $x \in L$
- And
  - probabilistic check of π_x reads 4i bits and has error probability $1/2^i$

- Server
  - Sends code
  - Computes prob check proof π_x
- Customer
  - Prob. selects 4i positions in π_x, sends
- Server
  - Sends 4i bits of π_x
- Customer
  - Checks π_x and accepts if proof is good

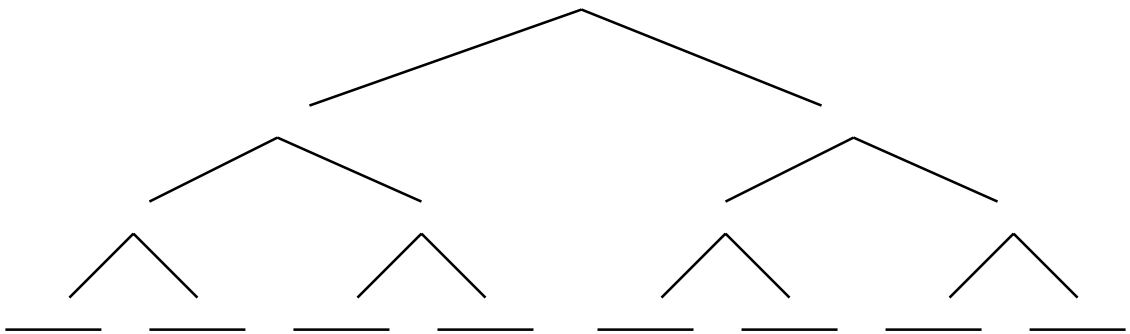
Big problem



- When does server compute π_x ?
- Customer accepts based on k (=4i) bits
  - doesn't know that b_1, b_2, ..., b_k come from π_x
    - if server knows Customer's algorithm, can try all possible combinations of k bits and win …

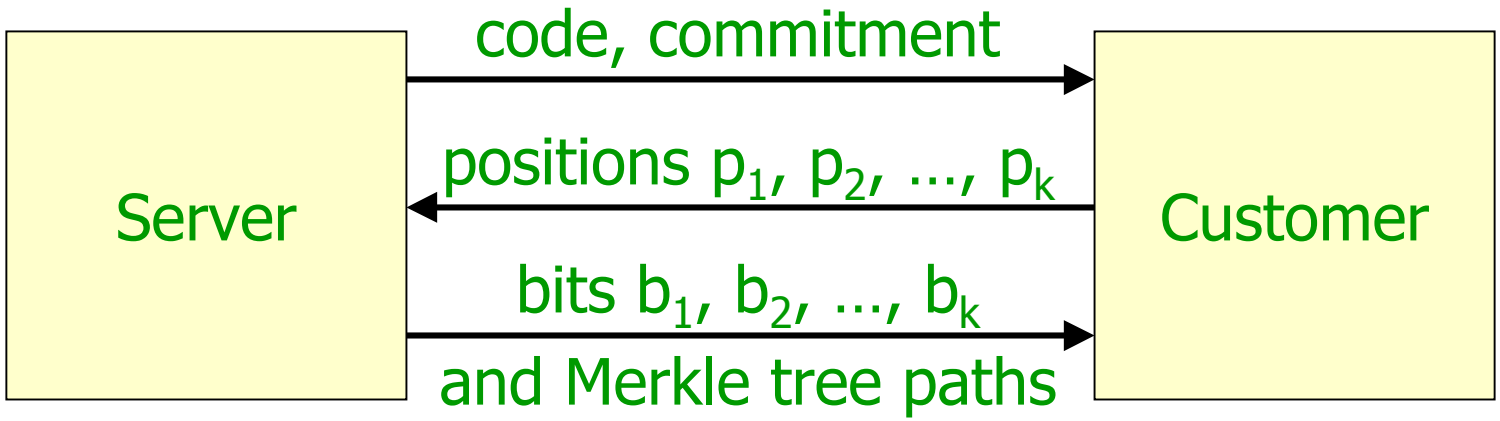
- Fair version
  - Server chooses scissors, rock, or paper
  - Client chooses scissors… and sends
  - Server determines winner and sends
- Unfair version
  - Server postpones choice
  - Client chooses scissors… and sends
  - Server chooses, determines winner …

- PCP works if server commits to proof
  - Consumer only looks at constant # of bits
- Commit to huge dataset
  - with only tiny fraction revealed
  - use Merkle trees



Protocol that works

Lincoln, Mitchell, Rubinfeld, Kannan, Dwork, Scedrov 01



- Server must compute π_x first
  - Commitment prevents cheating
- Customer accepts based on k (=4i) bits
  - Uses commitment to check that b_1, b_2, …, b_k come from π_x

What does this achieve?

- Server sends
  - Code (as standard PCC)
  - Commitment (small hash value)
- Customer sends
  - k positions of size log $|\pi_x|$ = log $|x|^n$
- Server sends
  - k paths of size log $|\pi_x| \bullet |$ hash value $|$
- Customer accepts
  - Possible error $1/2^i$ where i = k/4

So What?

- Mobile code can convince target server that a proof exists
- Even for very complicated properties, very big programs, very big proofs
- Using extremely small bandwidth
- At least theoretically

- Additional possibilities
  - testing-result-carrying code

Policy Analysis and Management

Data Security

Delegation of Trust

Merging of Policies

Data security policy is a key problem:

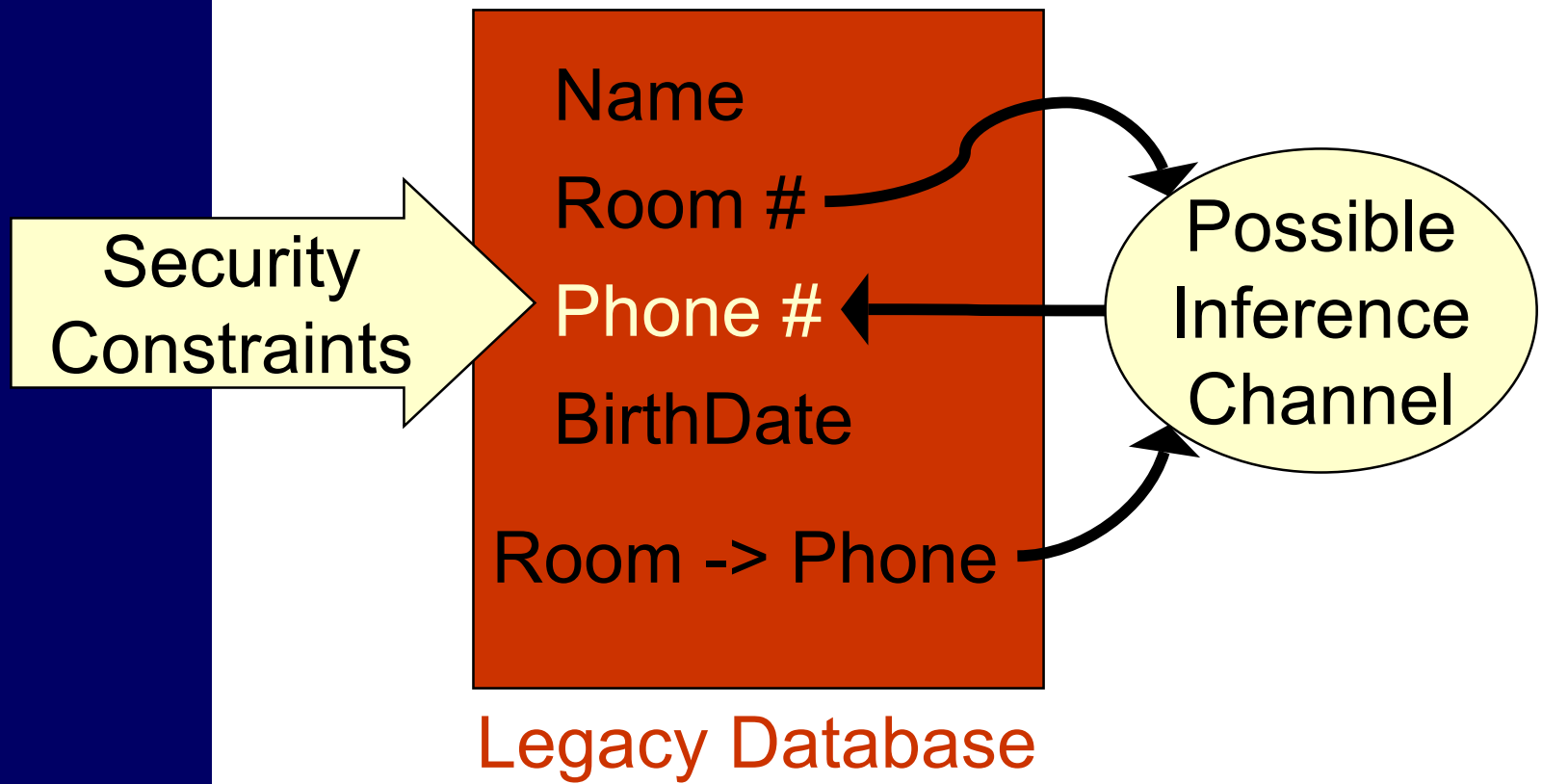
- User E is authorized to view/update X
- Policy says only senior-staff can see both salary and IP data
- Agent A delegates to agent B task of updating DB1 with new data from DB2

Given a partially specified data security policy, can it be completed consistently?
Given two data security policies, are they consistent? Can they be made consistent?

One Problem in Trust Management: Inference from Constraints

- Given legacy DB, constraints on certain data visibility, and simple inference rules, what is the minimum classification necessary for all elements?
- Thought to require exponential time

Example: Inference from Constraints



So if phone must be secret, then room, or room->phone map must be secret

More Precisely

Given lattice of security levels L1, L2, …

Public, Secret, Top Secret, MedicalA17

Given large set of elements E1, E2, …

Myphone#, F14wingspar, Testresult413

Given security constraints of the form:

E1 > L1

Battleplan14 > Top Secret

LUB(E1,E2) > E3

{Room#, Room->Phone} > PhoneBook

Given visibility constraints of the form:

E1 < L1

RealFiscalResults < CEO

GLB(E1,E2) > E3

More Precisely

Produce mapping from elements to lattice points
- Satisfying each constraint
- Maximal visibility (no strictly lower classification of elements satisfies all constraints)

Thought to be intractable (single exponential)
For fixed lattice can calculate in N^2 worst-case
On examples with large numbers of elements and large number of rules, implementation behaves linearly.

Dawson, Lincoln, Samarati

So What?

Benefits

Where is this useful?

Mobile code security policy. Agents can calculate their acceptance or rejection of execution environment security policies.

- Optimization
- Checking consistency
- Merging multiple policies

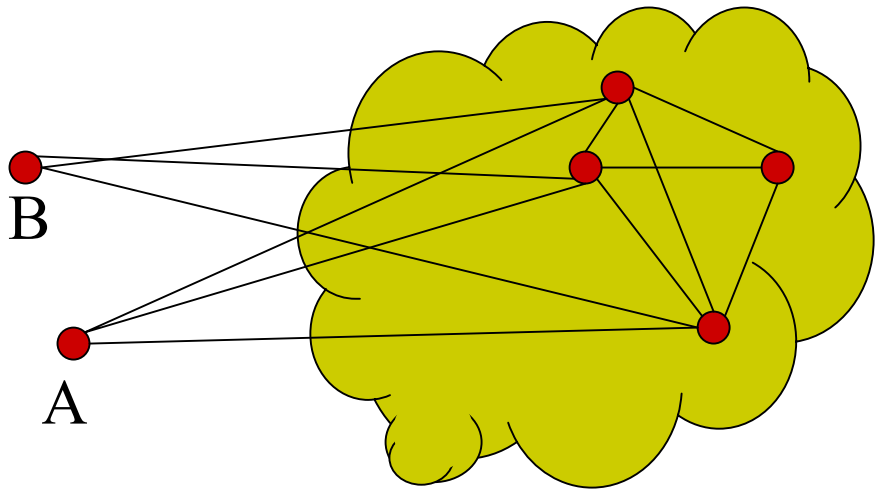
This is a disguised theorem proving problem. Look at the lattice ordering relation as consisting of logical inference rules

Agents Keeping Secrets

Can Agents Keep Secrets from their hosts?

General information assurance capabilities
For agent-based platforms

Networked secrets can be MORE SECURE than secrets kept on a single host



Secret Sharing Agents

Scaleable

Provably secure even when an agent is compromised

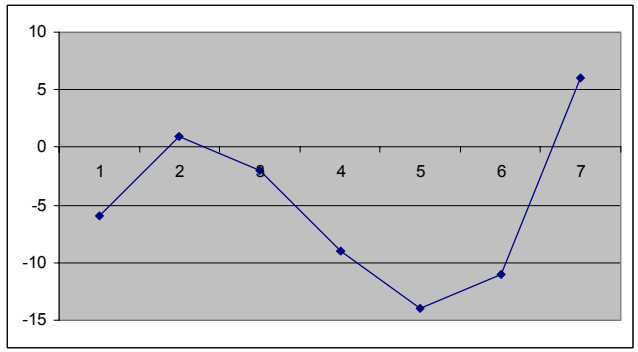
But not good enough: sequential compromise

Splitting a secret among multiple agents

Say you have 7 agents, and want to share a secret, say "-29"

Produce a random polynomial P(x) that takes value "-29" on zero. That is, P(0)= -29.

Send P(1) to agent 1, P(2) to agent 2, etc.



Proactive Security For Agents

Yung, et al

Scaleable

Provably secure even when agents are all sequentially compromised

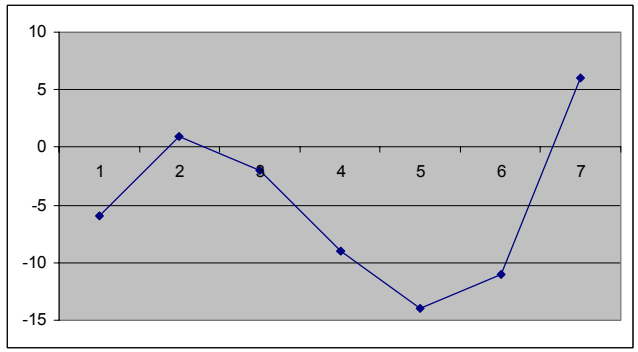
Efficient

Rotating the shares of the secret prevents sequential attackers gaining too much knowledge

Easy if there is a trusted central server

Can distribute the computation of new secret shares, but all agents must forget old shares

Define type of agents that forget



Platform for Secure Agent Interaction

General information assurance capabilities

Shared secrets can be used to sign digital documents without assembling the secret on any one host

Distributed rekeying of groups
- Change key without central server

Distributed group management
- Add and remove members

Summary

Agent Futures

Beneftis of formal methods approaches to mobile code

Agents can add new security risks

with modern cryptographic primitives and recent logic-based approaches to type systems, secure agent architecture is perhaps possible

Formal methods approaches and tools may provide value to mobile code applications

We are working on mobile code infrastructure based on Mobile Maude, a declarative reflective rewriting-logic based system

Outline

Formal Methods

Agents

Security Policies

What Next For Formal Methods?

More Automation

Less Intrusiveness (Invisible?)

More Expressive Types

Application to Mobile Code

Application to Biology
Symbolic Systems Biology