Language-Based Information-Flow Security

Andrei Sabelfeld Andrew C. Myers
Cornell University

survey appeared in
IEEE-JSAC, Jan. 2003

Dagstuhl
Oct. 2003

A scenario: free service software

Users freely download and use the software providing a service:

- Grokster, Kazaa, Morpheus,... are file-sharing services helping users exchange files
- Come with “hooks” for automatic updates
- Support advertisement to justify cost

Real story: malware

Users are tricked to download software bundled with:

- Homepage/search hijackers (MySearch)
- Unsolicited pop-up ads
- Rewriting URLs to override original ads with own
- “Hooks” for automatic updates are used to execute the advertiser’s arbitrary code (MediaUpdate, DownLoadware)
- Information gathering—visited URLs and filled forms are forwarded to a third-party (Gator, IPInsight, Transponder)

General problem: malicious and/or buggy code is a threat

- Trends in software
  - mobile code, executable content
  - platform-independence
  - extensibility
- These trends are attackers' opportunities!
  - easy to distribute worms, viruses, exploits,...
  - write (an attack) once, run everywhere
  - systems are vulnerable to undesirable modifications
- Need to keep the trends without compromising information security

Language-based security

- Looking under the street light...
  Attacker model:
  - eavesdropping on network
  - modifying network traffic
  - trusted communication endpoints

$\Rightarrow$ cryptographic protection of communication

- ...for a key that lies somewhere else!
  Real story [CERT]: Most attacks are
  - remote penetrations (buffer overruns, format strings, RPC vulnerabilities,...)
  - malware (viruses, worms, DDoS slaves,...)

$\Rightarrow$ need protection at application level

Information security: confidentiality

- Confidentiality: sensitive information must not be leaked by computation (non-example: spyware attacks)
- End-to-end confidentiality: there is no insecure information flow through the system
- Standard security mechanisms provide no end-to-end guarantees
  - Security policies too low-level (legacy of OS-based security mechanisms)
  - Programs treated as black boxes

Confidentiality: standard security mechanisms

Access control

+prevents “unauthorized” release of information

- but what process should be authorized?

Firewalls

+permit selected communication

- permitted communication might be harmful

Encryption

+secures a communication channel

- even if properly used, endpoints of communication may leak data

Confidentiality: standard security mechanisms

Antivirus scanning

+rejects a "black list" of known attacks

- but doesn't prevent new attacks

Digital signatures

+help identify code producer

-no security policy or security proof guaranteed

Sandboxing/OS-based monitoring

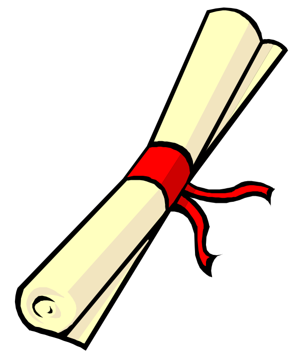
+good for low-level events (such as read a file)

-programs treated as black boxes

$\Rightarrow$ Useful building blocks but no end-to-end security guarantee

Confidentiality: language-based approach

- Counter application-level attacks at the level of a programming language—look inside the black box! Immediate benefits:
- Semantics-based security specification
  - End-to-end security policies
  - Powerful techniques for reasoning about semantics



- Static security analysis
  - Analysis enforcing end-to-end security
  - Track information flow via security types
  - Type checking by the compiler removes run-time overhead

Dynamic security enforcement

Java's sandbox, OS-based monitoring, and Mandatory Access Control dynamically enforce security policies; But:

high(secret)

low(public)

```
h:=…;
l:=false;
if h then l:=true
     else skip
```

implicit flow from h to l

Problem: monitoring a single execution path is not enough!

Static certification

- Only run programs which can be statically verified as secure before running them
- Static certification for inclusion in a compiler [Denning & Denning'77]
- More precise implicit flow analysis
- Enforcement by static analysis (e.g., security-type systems)

A security-type system

Expressions:

exp : high

$$\frac{h \notin Vars(exp)}{exp : low}$$

Atomic commands (pc represents context):

[pc] ⊢ skip

[pc] ⊢ h:=exp

$$\frac{exp : low}{[low] \vdash l := exp}$$

context

A security-type system: Compositional rules

$$\frac{[\text{high}] \vdash C}{[\text{low}] \vdash C}$$

$$\frac{[pc] \vdash C_1 \quad [pc] \vdash C_2}{[pc] \vdash C_1; C_2}$$

implicit flows: branches of a high if must be typable in a high context

$$\frac{exp{:}pc \quad [pc] \vdash C_1 \quad [pc] \vdash C_2}{[pc] \vdash \text{if } exp \text{ then } C_1 \text{ else } C_2}$$

$$\frac{exp{:}pc \quad [pc] \vdash C}{[pc] \vdash \text{while } exp \text{ do } C}$$

A security-type system: Examples

$[low] \vdash h{:=}l{+}4;\ l{:=}l{-}5$

$[pc] \vdash \text{if } h \text{ then } h{:=}h{+}7 \text{ else skip}$

$[low] \vdash \text{while } l{<}34 \text{ do } l{:=}l{+}1$

$[pc] \nvdash \text{while } h{<}4 \text{ do } l{:=}l{+}1$

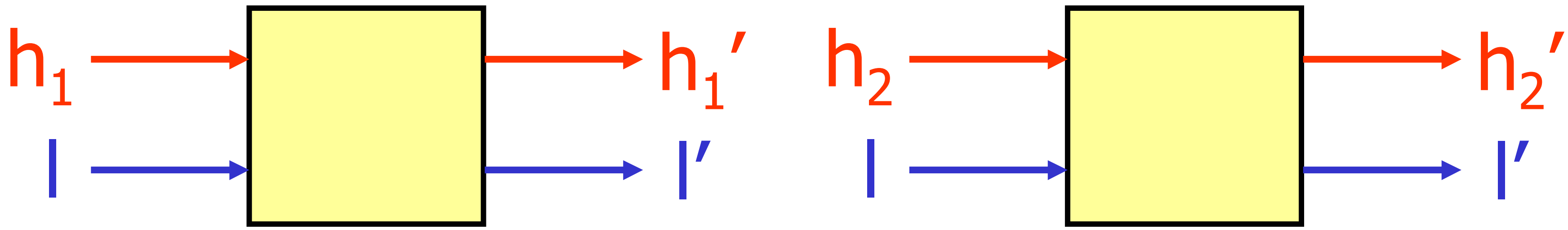
Semantics-based security

- What end-to-end policy such a type system guarantees (if any)?
- Semantics-based specification of information-flow security [Cohen'77], generally known as noninterference [Goguen & Meseguer'82]:

> A program is secure iff high inputs do not interfere with low-level view of the system

Semantics-based security

- Noninterference [Goguen & Meseguer]: as high input varied, low-level outputs unchanged



- Semantics-based security for C:

$$\forall mem, mem'.\ mem =_L mem' \Rightarrow [\![C]\!] mem \approx_L [\![C]\!] mem'$$

Low-memory equality: $(h,l) =_L (h',l')$ iff $l=l'$

C's behavior: semantics $[\![C]\!]$

Low view $\approx_L$: indistinguishability by attacker

Semantics-based security

- What is $\approx_L$ for our language?
- Intention: $[pc] \vdash C \Rightarrow C$ is secure
  I.e., if C is typable then

$$\forall s_1, s_2.\ s_1 =_L s_2$$
$$\Rightarrow [\![C]\!]s_1 \approx_L [\![C]\!]s_2$$
$$\Leftrightarrow [\![C]\!]s_1 \neq \bot \neq [\![C]\!]s_2 \Rightarrow [\![C]\!]s_1 =_L [\![C]\!]s_2$$

Termination-insensitive interpretation of $\approx_L$

Evolution of language-based information flow

Before mid nineties two separate lines of work:

Static certification, e.g., [Denning & Denning'76, Bergeretti & Carré'85, Mizuno & Oldehoeft'87, Palsberg & Ørbæk'95]

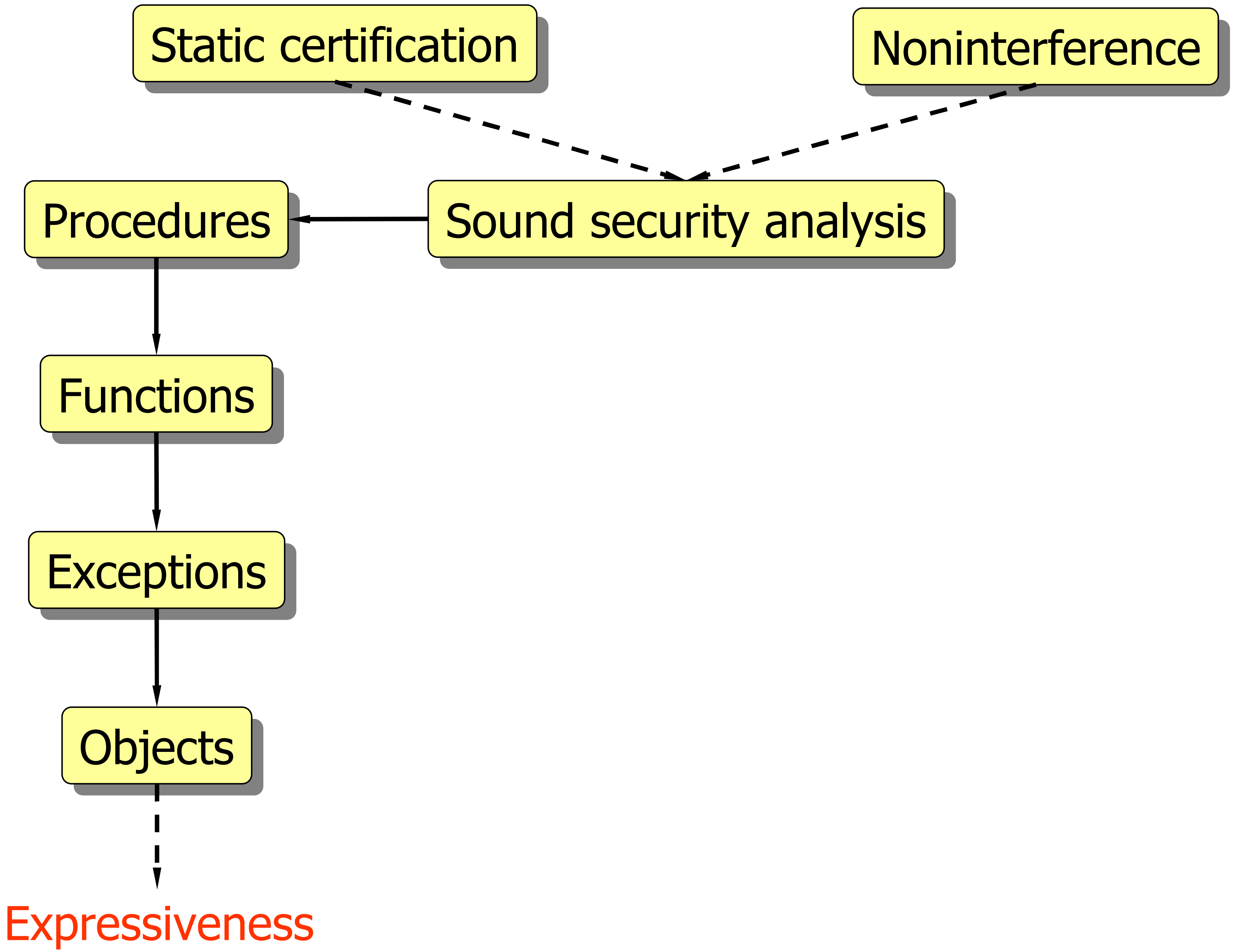
Security specification, e.g., [Cohen'77, Andrews & Reitman'80, Banâtre & Bryce'93, McLean'94]

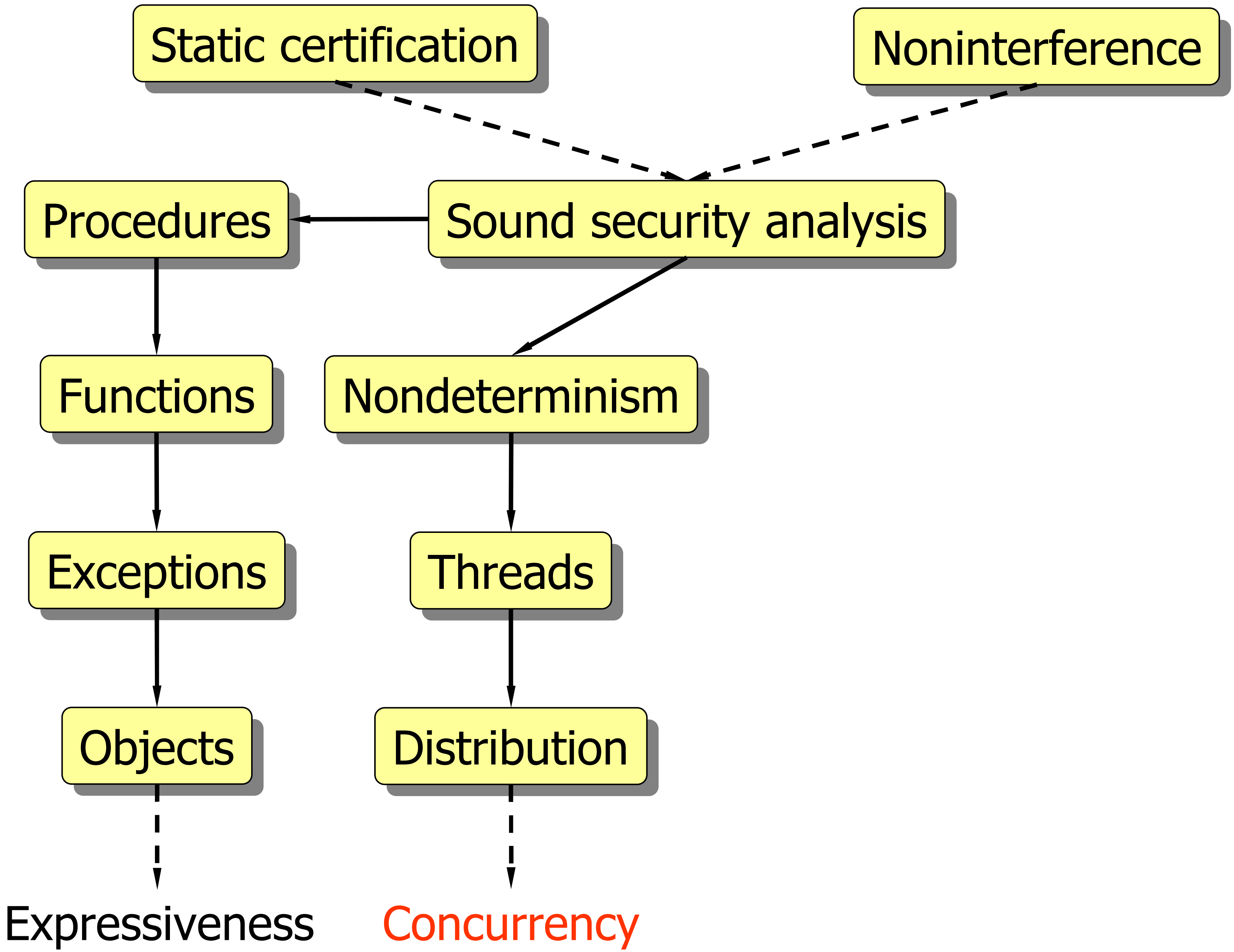
Volpano et al.'96: First connection between noninterference and static certification: security-type system that enforces noninterference

Evolution of language-based information flow

Four main categories of current information-flow security research:

- Enriching language expressiveness
- Exploring impact of concurrency
- Analyzing covert channels (mechanisms not intended for information transfer)
- Refining security policies


Static certification
Noninterference
Procedures
Sound security analysis
Functions
Exceptions
Objects
Expressiveness


Static certification
Noninterference
Procedures
Sound security analysis
Functions
Nondeterminism
Exceptions
Threads
Objects
Distribution
Expressiveness
Concurrency

Concurrency: Nondeterminism

- Possibilistic security: variation of h should not affect the set of possible l
- An elegant equational security characterization [Leino & Joshi'00]: suppose HH ("havoc on h") sets h to an arbitrary value; C is secure iff

$$\forall s. [\![HH; C; HH]\!] s \approx [\![C; HH]\!] s$$

Concurrency: Multi-threading

- The high data must be protected at all times: h:=0; l:=h is secure as a sequential program, but not when h:=h' is run in parallel
- A type system [Smith & Volpano'98] for nondeterministically scheduled threads rejects high while loops, but not leaks via schedulers:

```
if h then sleep(100);
l:=1
```

||

```
sleep(50); l:=0
```

- Encoding of a timing leak to a direct leak

Concurrency: Multi-threading

- A later work [Volpano & Smith'98] proposes a "protect" command for wrapping high ifs
- Scheduler-independent security; no need for "protect" via Agat's transformation [Sabelfeld & Sands'00]
- Thread synchronization (as by semaphores) may lead to leaks by blocking [Sabelfeld'01]
- Permissive type systems for multithreaded programs [Boudol & Castellani'01,'02]
- A uniform type system [Honda et al.'00,'02] and a light type system [Pottier'02] for noninterference in π–calculus
- Security through low determinism [Zdancewic & Myers'03]

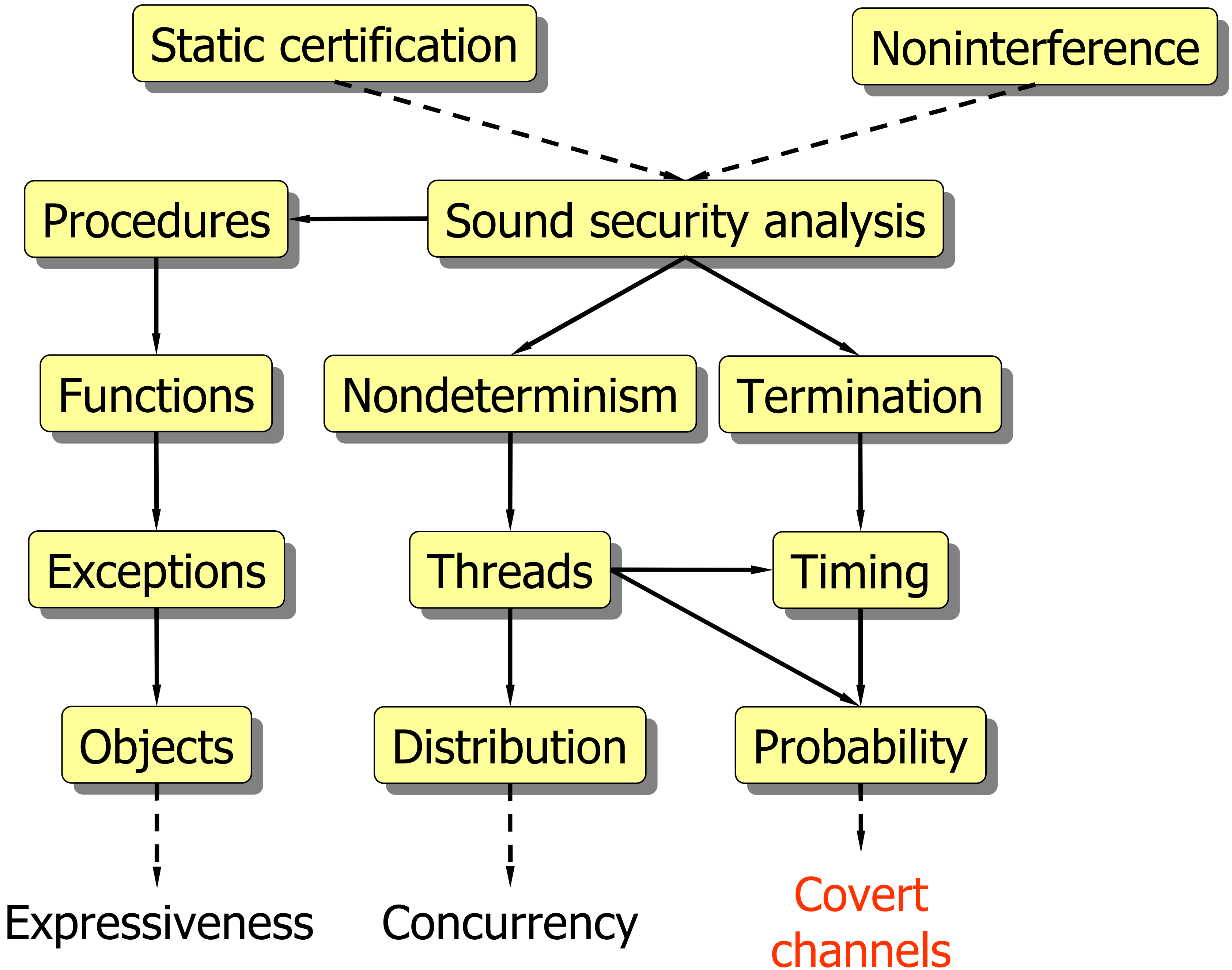
Confidentiality issues for distributed systems

concur-rency
- Blocking of a process observable by other processes (also timing, probabilities,...)

distri-bution
- Messages travel over publicly observable medium; encryption protects messages' contents but not their presence
- Mutual distrust of components
- Components (hosts) may be compromised/ subverted; messages may be delayed/lost

Concurrency: Distribution

- Jif/split: An architecture for secure program splitting to run on heterogeneously trusted hosts [Zdancewic et al.'01]
- Type systems for secrecy for cryptographic protocols in spi-calculus [Abadi'97, Abadi & Blanchet'01]
- Logical relations for the low view [Sumii & Pierce'01]
- Interplay between communication primitives and types of channels [Sabelfeld & Mantel'02]
- Secure replication and partitioning [Zheng et al.'03]



Static certification
Noninterference
Procedures
Sound security analysis
Functions
Nondeterminism
Termination
Exceptions
Threads
Timing
Objects
Distribution
Probability
Expressiveness
Concurrency
Covert channels

Covert channels: Termination

- Covert channels are mechanisms not intended for information transfer

> Is while h>0 do h:=h+1 secure?

- Low view $\approx_L$ must match observational power (if the attacker observes (non)termination):

$$s \approx_L s' \text{ iff } s = \bot = s' \vee (s \neq \bot \neq s' \wedge s =_L s')$$

- PER model can be naturally lifted to handle termination

Covert channels: Timing

- Nontermination $\approx_L$ time-consuming computation
- Bisimulation-based $\approx_L$ accurately expresses the observational power [Sabelfeld & Sands'00, Smith'01,'03]
- Agat's cross-copying technique for transforming out timing leaks [Agat'00]

Covert channels: Probabilistic

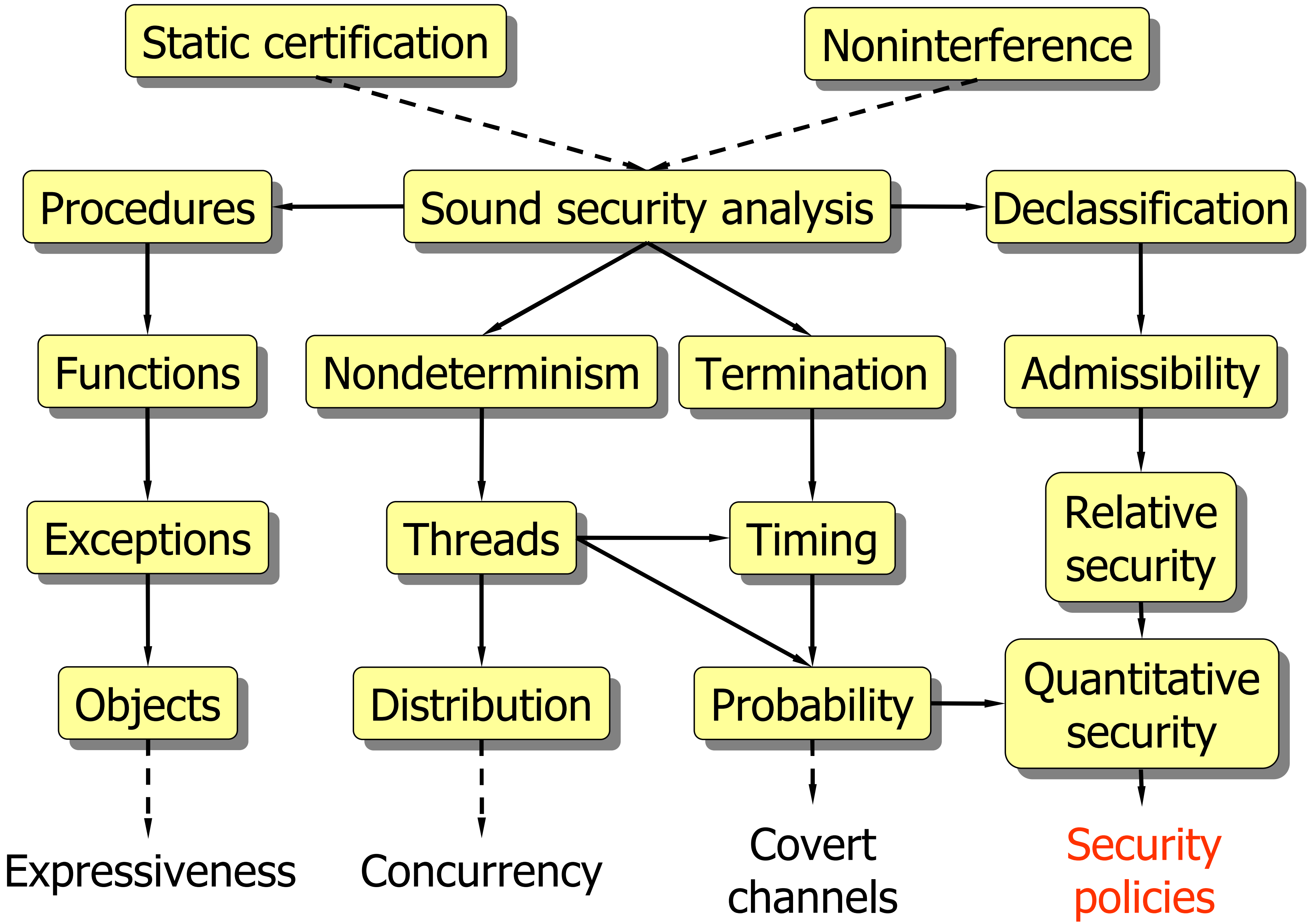
- Possibilistically but not probabilistically secure:

```
if h then sleep(100);
l:=1
```

||

```
sleep(50); l:=0
```

- Probability-sensitive $\approx_L$ by PERs [Sabelfeld & Sands'99]
- Probabilistic bisimulation-based security [Volpano & Smith'99, Sabelfeld & Sands'00, Smith'01,'03]



Static certification
Noninterference
Procedures
Sound security analysis
Declassification
Functions
Nondeterminism
Termination
Admissibility
Exceptions
Threads
Timing
Relative security
Objects
Distribution
Probability
Quantitative security
Expressiveness
Concurrency
Covert channels
Security policies

Security policies

- Many programs intentionally release information, or perform declassification
- Noninterference is restrictive for declassification
  - Encryption
  - Password checking
  - Spreadsheet computation (e.g., tax preparation)
  - Database query (e.g., average salary)
  - Information purchase
- Most approaches to information flow control ignore declassification—need more flexible security policies

Security policies: Declassification

- To legitimize declassification we could add to the type system:

declassify(h) : low

- But this violates noninterference
- What's the right typing rule? What's the security condition that allows intended declassifications?

Security policies

- Secrecy in protocols [Abadi'97]
- Relative secrecy [Volpano&Smith'00, Volpano'00]
- Quantitative security [Denning'82,Clark et al.'02,Lowe'02]
- Approximate security $(\approx_L)_\varepsilon$ [Di Pierro et al.'02]
- Complexity-theoretic security [Laud'01,'03]
- Admissibility [Dam & Giambiagi'00, Giambiagi & Dam'03]
- Decentralized security model [Myers&Liskov'97]
- Robust declassification [Zdancewic&Myers'01, Zdancewic'03]
- Access control policies for secure information flow [Banerjee & Naumann'03]
- Cryptographic types [Duggan'02]
- Type-based distributed access control [Chothia et al.'03]

Language-based information security: challenges

Some essential challenges—some are not addressed by current trends!

- ▷ • System-wide security
- ▷ • Certifying compilation
- ▷ • Attacks beyond abstraction
- ▷ • Dynamic policies
- ▷ • Practical issues

$\Rightarrow$ Opportunities for integrating model checking, logic, theorem proving, code rewriting,…

Conclusion

- Security practices not capable of tracking information flow
- Language-based security: effective information flow security models (semantics-based security) and enforcement mechanisms (security-type systems)
- Progress on expressive languages, concurrency, covert channels, security policies
- Critical challenges remain for language-based mechanisms to become a part of security practice

End of talk