東京大学2003年度COEシンポジウム

<文部科学省科学研究補助金　特定領域研究(H12-H15)>

社会基盤としての セキュアコンピューティングの 実現方式の研究

http://anzen.is.titech.ac.jp

代表者 米澤 明憲

東京大学大学院

情報理工学系研究科コンピュータ科学専攻

文部科学省科学究費　特定領域研究

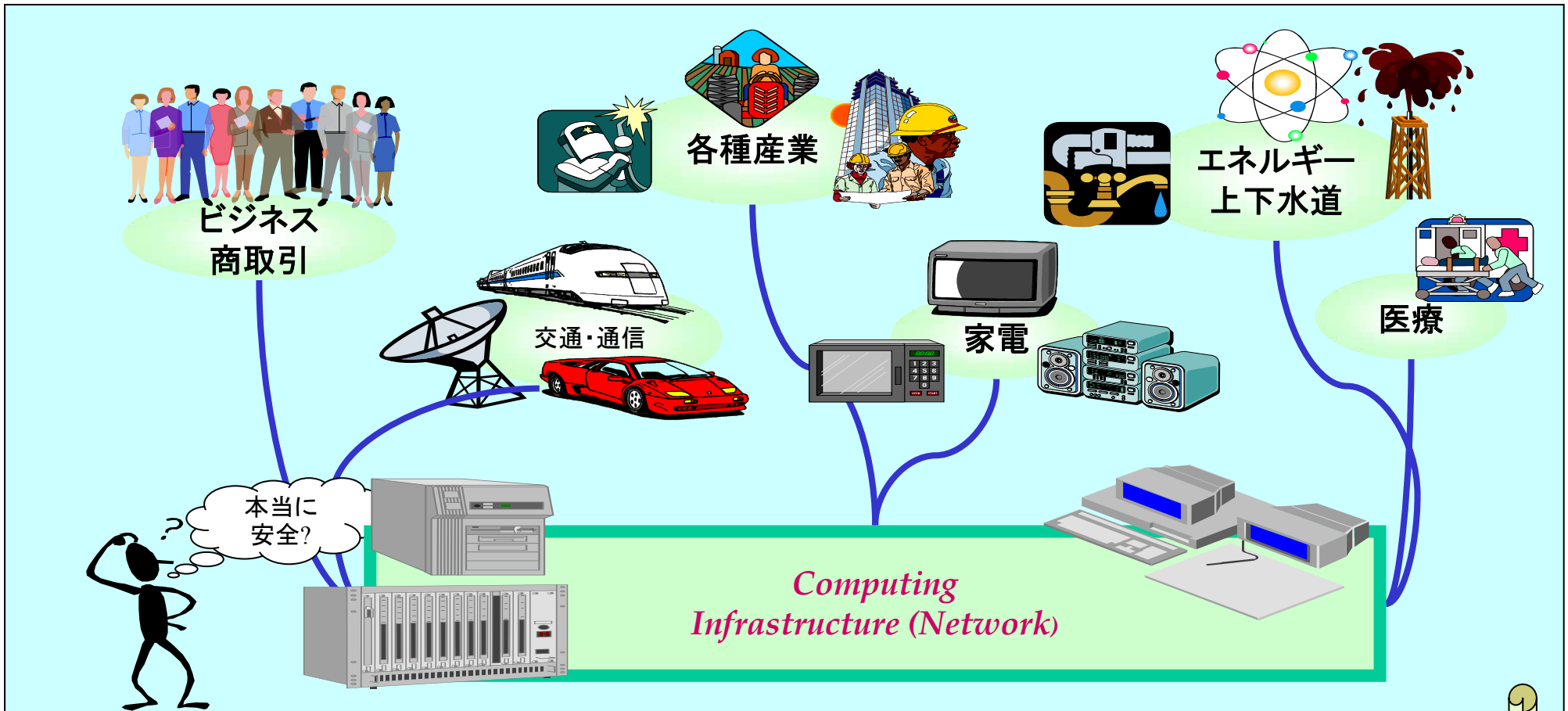
- 国として重点的に推進する研究課題
- 大学の研究者グループから組織（20人−50人）
- 研究提案を公募・審査
  - 採択率　15／150（毎年）
  - 分野　科学全分野一括　（「情報」は年に1，2件採択）
- 年間補助額金額　　5000万から3億円
- 期間：　2年--5年間
- 評価：　中間評価＋終了評価あり

我々の「研究領域」

- 研究組織:
  - 6つの大学にまたがる9つの研究グループ（約35名）
  - 助言・内部評価のために1統括グループ
- 期間: H12．9－H16．3
- 研究費: 年額1．3億円
- 研究対象: ソフトウエアセキュリティ

（暗号系は含まず）

社会基盤としてのコンピューティングインフラ



コンピュータとネットワークは，現代社会の「脳」であり「神経」でもある．社会の隅々にまで浸透し，人命，財産，治安をあずかるまでにいたったが，さまざまな危険にさらされているのが現実である．

我々の研究対象は

インターネットを構成するサーバコンピュータ
(e.g., WEBサーバ, メールサーバ, 計算サーバ)の

ソフトウエアシステムの安全保証

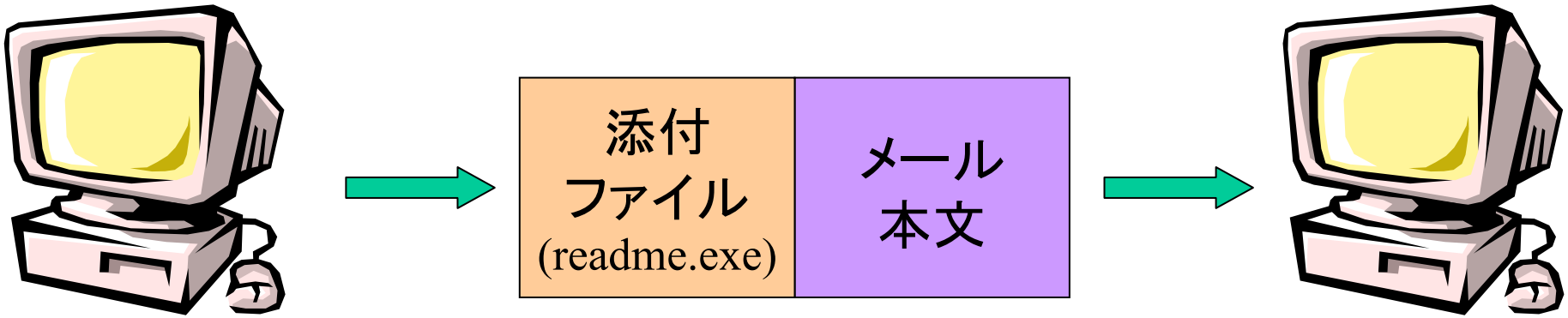
本研究が対処する脅威

- 悪意や誤りのあるプログラムが
  - 計算機システムを破壊（DOS攻撃を含む）
  - 秘密を漏洩
- コンピュータウィルスが
  - サーバプログラムの欠陥をついて侵入
- なりすまし（impersonation）
  - 正当な利用者のふりをして
    計算機システムを不正使用

ソフトウェアセキュリティ

- 攻撃に耐える
  - **E.g.** インターネットを使った直接攻撃
  - **E.g.** 添付ファイルによる間接攻撃
  - **E.g.** 同一コンピュータ内からの攻撃
- 事故に耐える
  - **E.g.** ハードウェア故障やネットワーク切断などの不慮の事故
- 頑強なソフトウェアをめざす
  - 開放性や接続性を保持
  - 攻撃や事故に耐えるソフトウェアの設計と実装

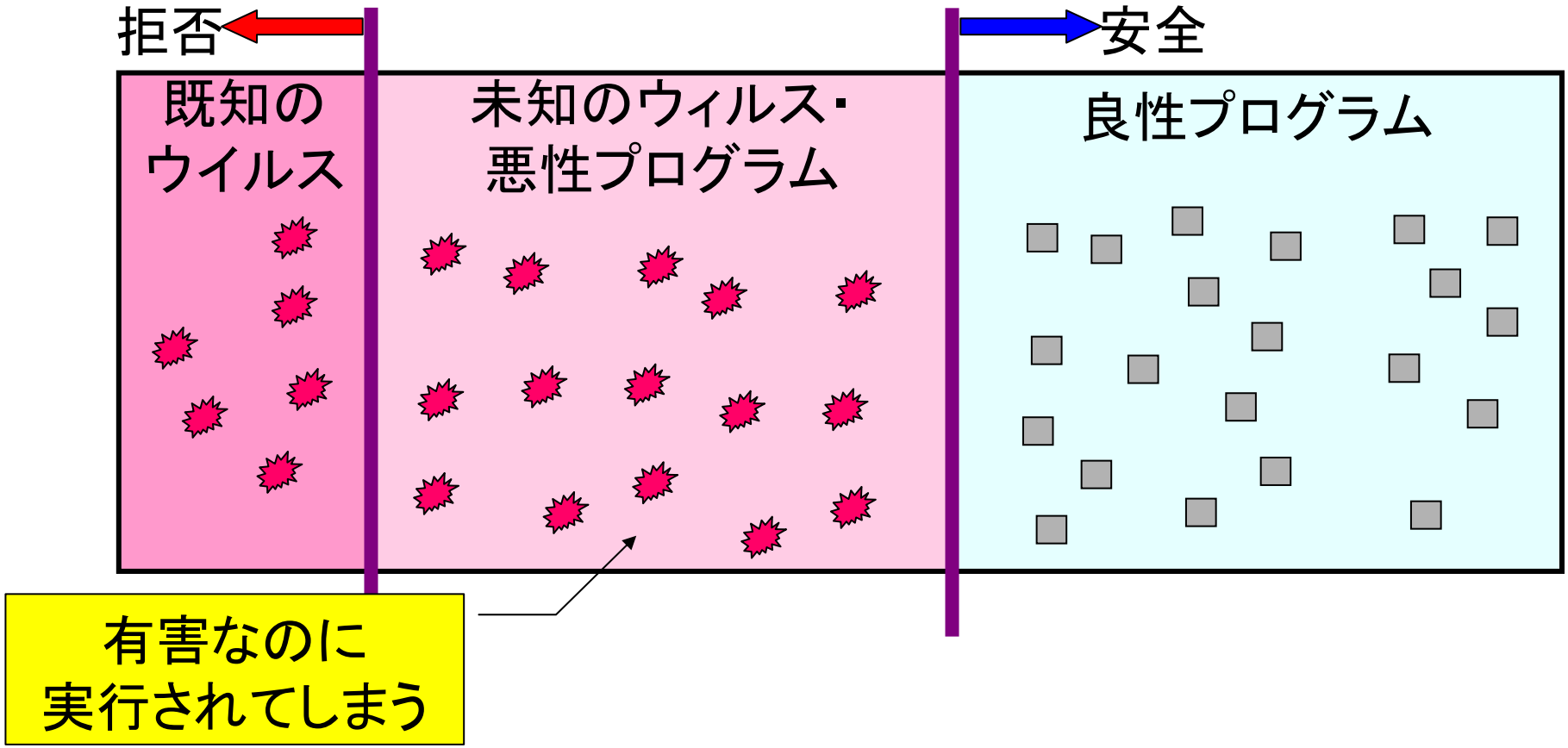
よくある攻撃の例:
メールに添付されたウイルス



- なぜメールシステムが攻撃されるか？
  - 郵便受けは，外に向かって常に開いている
  - 通常のメールシステムは悪質な添付ファイルの存在を想定していない
  - 「ユーザが判断できる」という逃げの姿勢で設計されているシステムが多い

従来の対策法

- 添付ファイルを開かないようにする。
- ワクチンで既知ウイルスを排除する。



我々の不正コード防御戦略

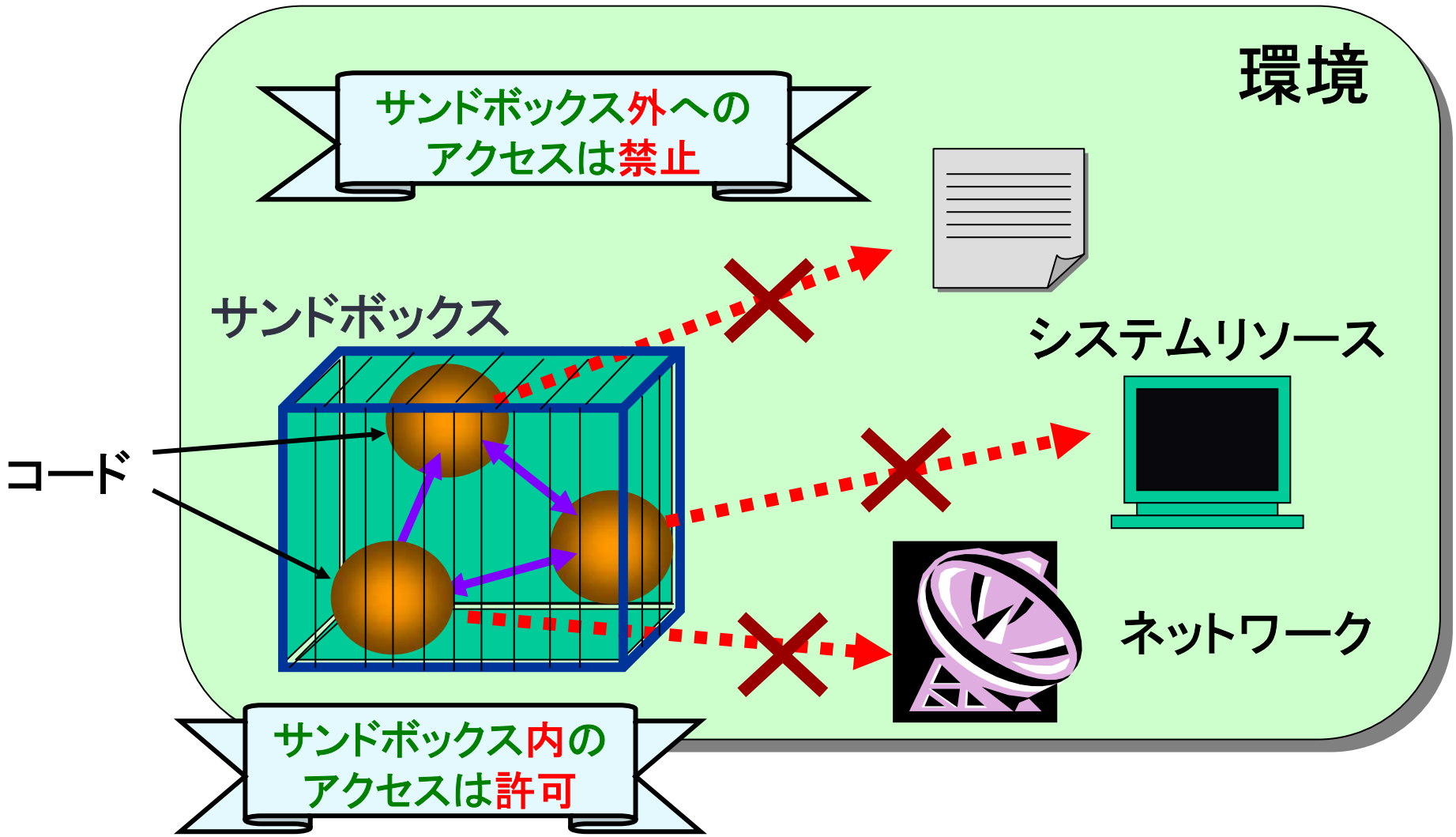
不正コード＝（ウイルスもその例）

1. 添付ファイル（コード）を理論に裏付けられた方法で解析し，危険性の有無を検査

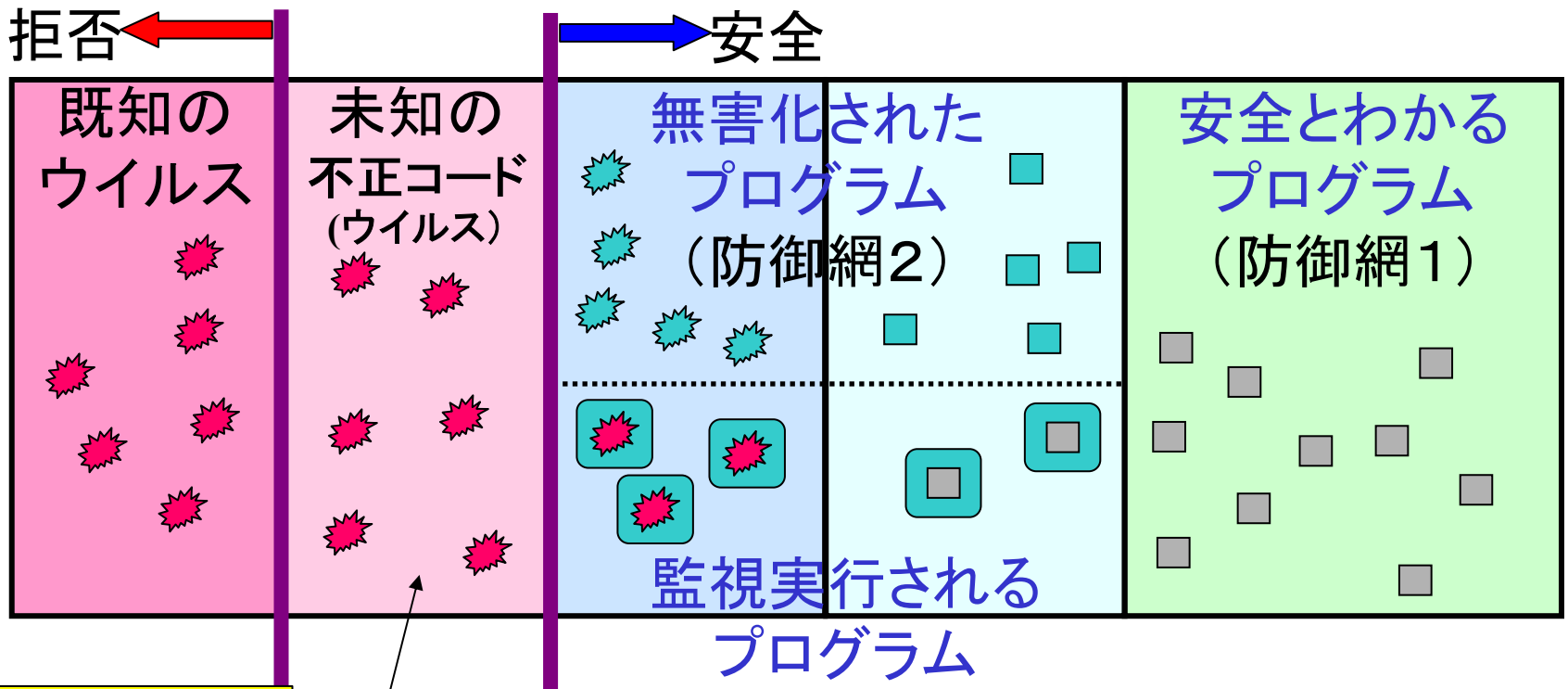
2. 危険性のある箇所は，安全なものに書き換える（無害化）
3. 以上ができない場合，OS（ or 仮想機械）で監視しながら実行

サンドボックス法

サンドボックス法（Sandboxing）
―仮想実行・監視実行―

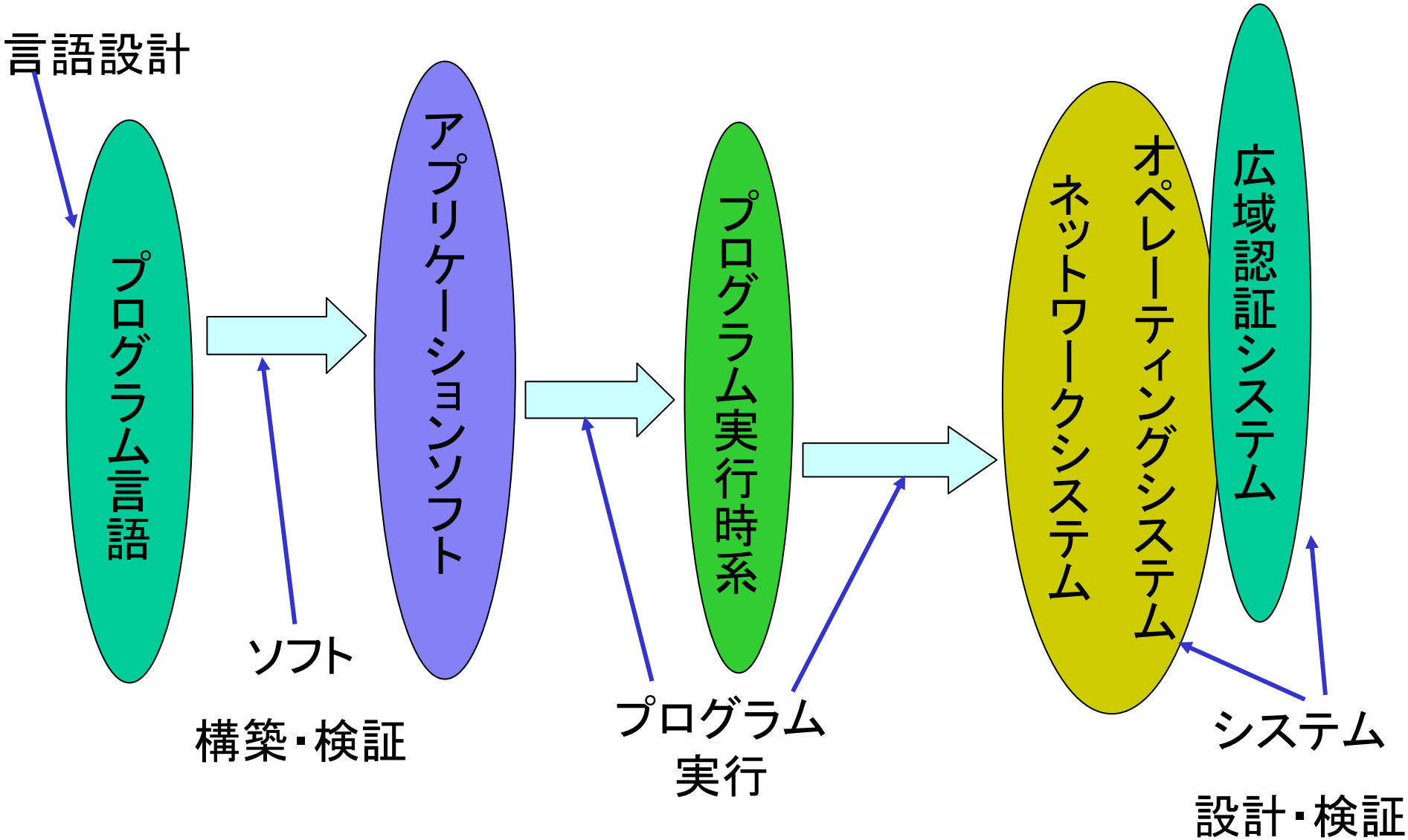


我々の防御網



この領域を
我々の研究で
できるだけ
小さくする

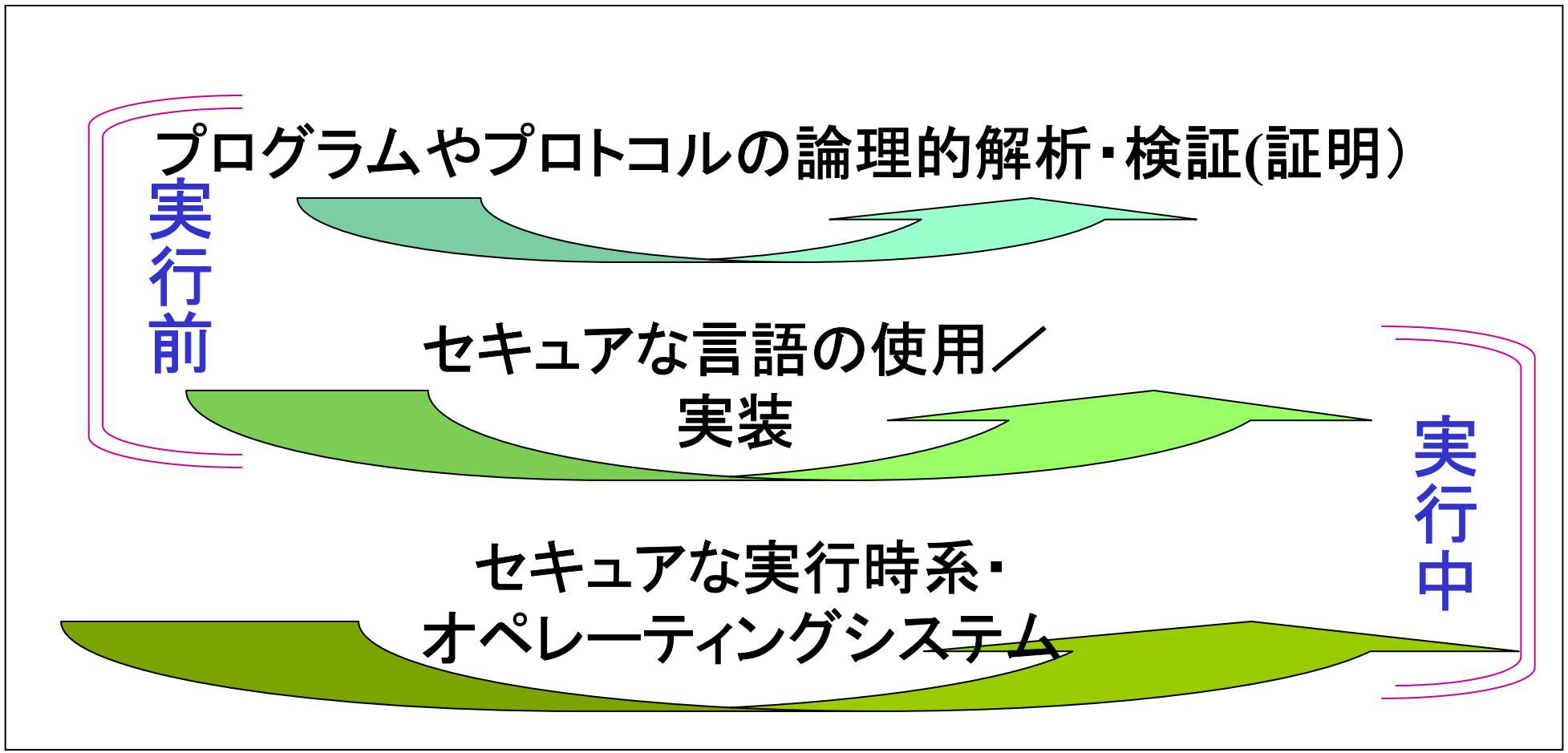
情報インフラの構成要素
―システムの構築・実行過程―



３重のセーフティーネット

- 理論的な検証・解析手法
  - プログラム実行前に危険性を検出
  - システム設計時に安全性を証明
  - ユーザや計算機の認証のためのプロトコルの安全性の証明
- プログラミング言語・記述系
  - 安全性を示しやすく、かつ効率の良い
    プログラムの作成を支援
- オペレーティングシステムなどの実行系
  - プログラミング言語・記述系では捕捉できない脅威を防止

我々の防御戦略
−3重のセーフティネット−



研究グループ（3重の安全網にほぼ対応）

A01（理論的解析・検証）

グループ代表者　米崎、二木、萩谷、小林

A02（言語・記述系）

グループ代表者　米澤、柴山、渡部

A03（OS・インフラ系）

グループ代表者　徳田、加藤、溝口

6つの大学：

慶応義塾大学、東京大学、東京工業大学

東京理科大学、筑波大学、北陸先端大学院大学

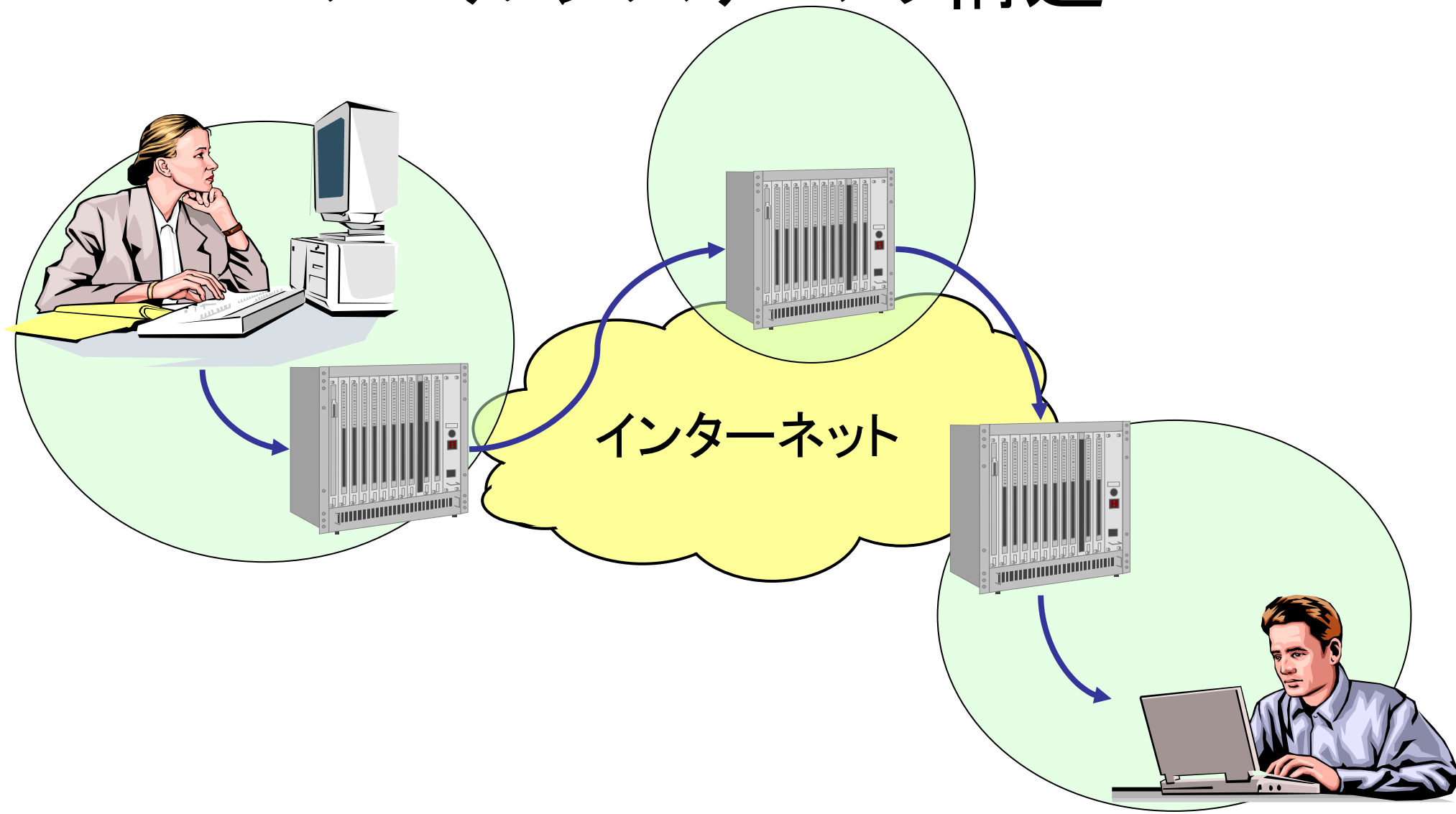
グループ間の連携研究のよる成果：

安全なメールシステム
「AnZenメール」
の構築

メールシステム構築の目的

- 安全なソフトウェアシステム構築方式を体系化
  - 想定する攻撃は，ワーム，ウイルス，クラッキングなど
  - ソフトウェア技術でソフトウェアを守る
  - 人的問題，社会的問題にはあまり踏み込まない
  - 暗号の問題にもあまり踏み込まない
- 領域に関連した技術の統合と実証試験
- そのための例題としてメールシステムを選択

メールシステムの構造

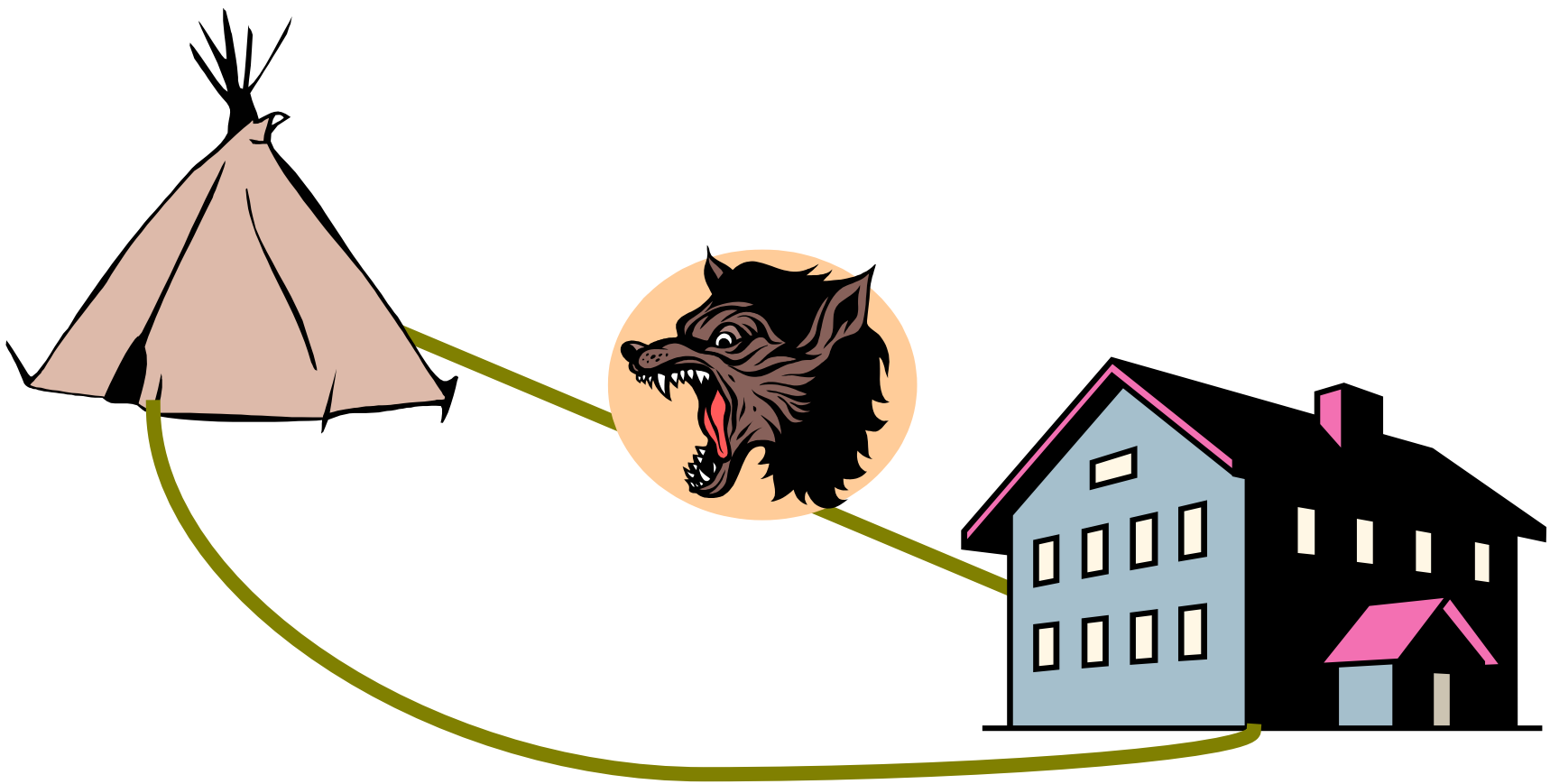


メールシステムの特徴

- 構成要素
  - 通信路
  - サーバ
  - クライアント（メーラ）
- セキュリティ保持の難しさ
  - 公開サーバ
  - 独自のルーティング
  - 公開アドレス
- 全世界規模での利用

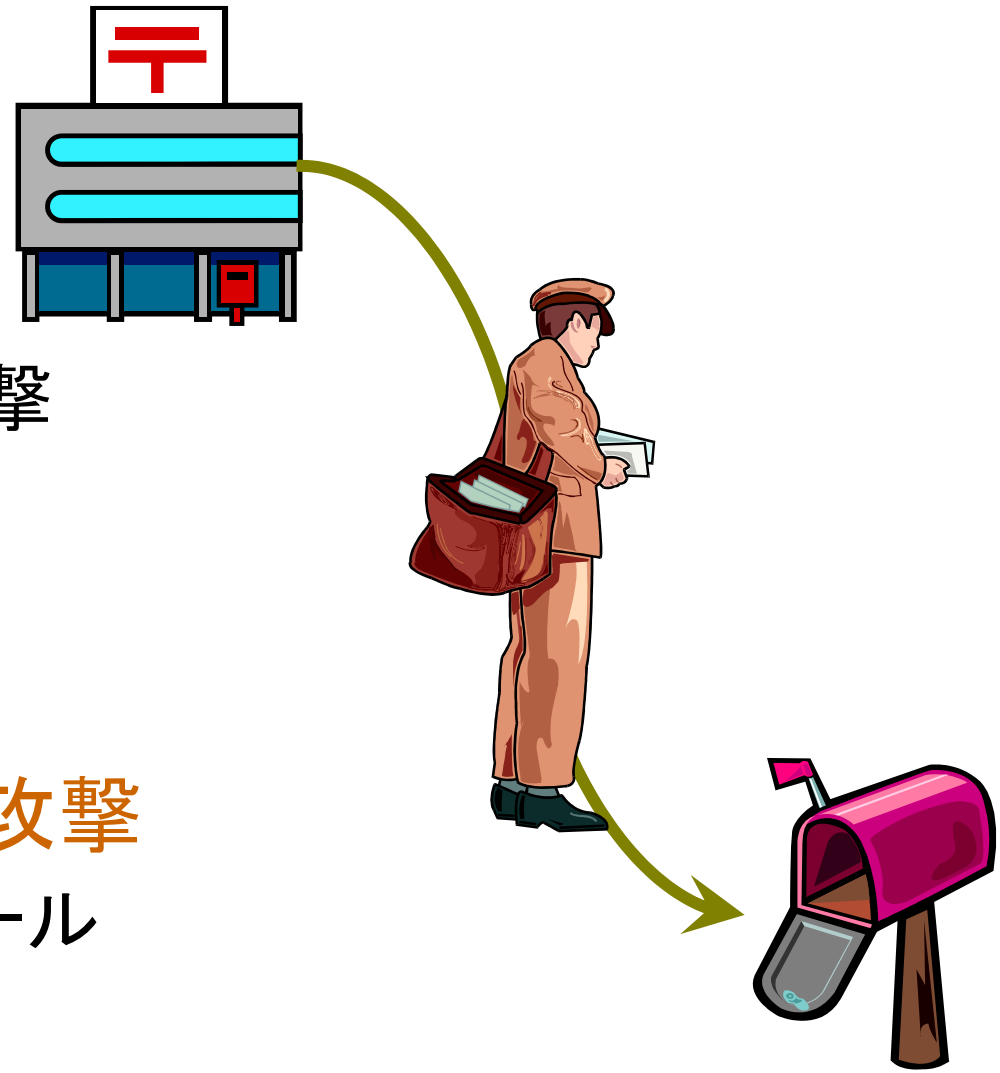
通信路の安全性

- 基本的には，暗号と冗長化で対応可能



メールシステムへの攻撃

- サーバへの攻撃
  - クラッキング
  - ワーム
  - サービス拒否攻撃
- 通信路への攻撃
  - 盗聴
  - 改竄
- クライアントへの攻撃
  - ウイルス添付メール
  - 迷惑メール

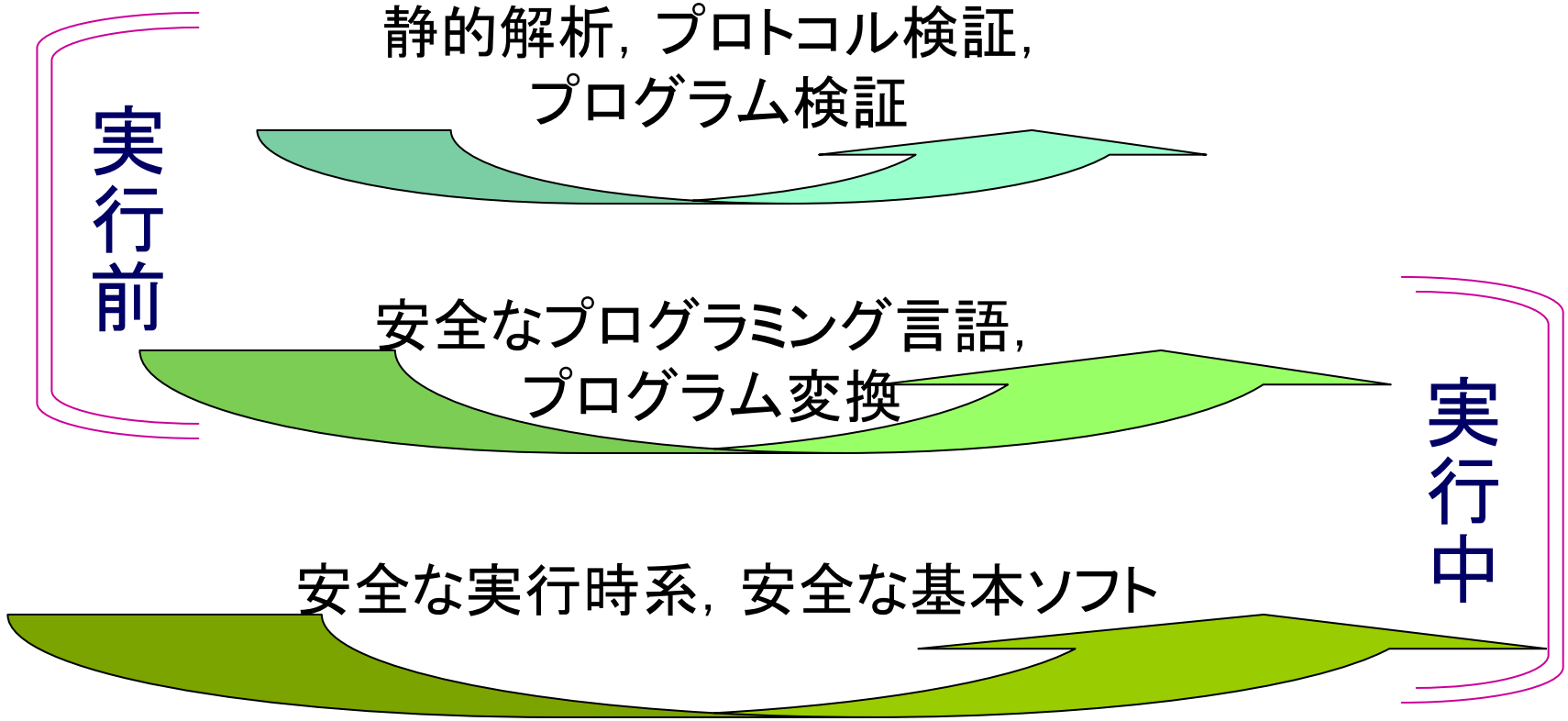


ユーザが起こす問題

- 社会的問題
  - 嘘や問題発言を含むメール
  - 社内情報やプライバシーの漏洩
  - 送信の否認，受信の否認

アプローチ (1/3)

- 3階層のセーフティネット



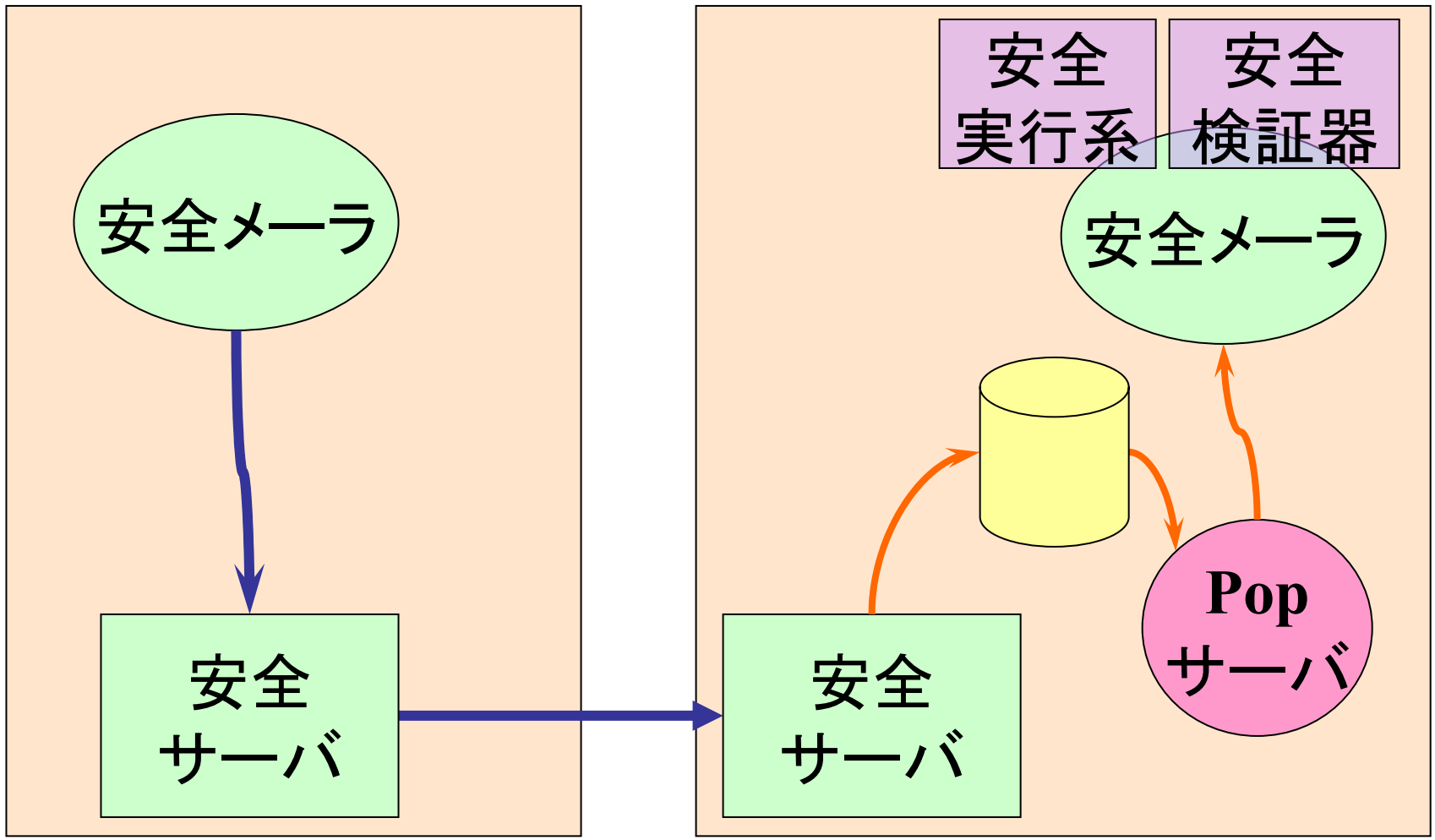
アプローチ (2/3)

- 検証＆解析
  - コストが高く，適用範囲もやや狭い
  - 実行性能に悪影響を与えない
- 安全な言語
  - ソフトウェアの設計・実装にある程度枠をはめる
  - バグや攻撃への耐性を高める
- 安全な実行時系
  - 実行時オーバーヘッドが発生
  - 適用範囲が広い

アプローチ (3/3)

- 三要素に分割し，各々に適した手法を適用
  - 安全プロトコル
  - 安全サーバ
  - 安全メーラ
- 単純かつセンシティブなものほど検証を強化

安全メールの構成



安全メールの基本機能

- 既存メールシステムとの互換性
  - **SMTP/POP3** をサポート
- 安全サーバ
  - （機能的には普通の）**SMTP** サーバ
- 安全メーラ
  - **MIME** のサポート
  - **Jar** ファイルの実行環境

安全メールの特徴 (1/2)

- 安全プロトコル
  - 改竄防止と経路認証が可能な配送プロトコル
  - 既存 **SMTP** サーバからは透過的
  - **BAN**論理を用いた理論的な検証
- 安全サーバ
  - メモリセーフな実装
  - **Coq** を用いた形式的かつ機械的な検証

安全メールの特徴 (2/2)

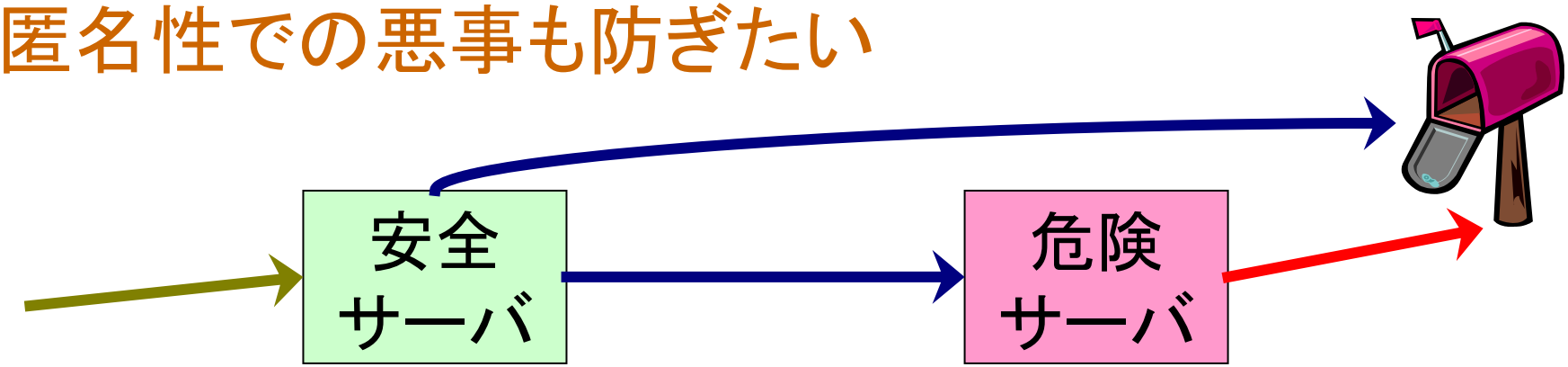
- 安全メーラ
  - 検証するには複雑すぎる
    - **JavaMail, Java Swing などに依存**
  - センシティブなのは添付ファイル実行部分
  - 検査器用のプラグインインタフェース
    - **E.g., 静的解析モジュール**
    - **E.g., コード変換モジュール**
    - **E.g., 動的モニタリングモジュール**
  - 実行器用のプラグインインタフェース

実装の状況

- **Java** 言語でメモリセーフに実装
- 基本機能と特徴的機能の一部が動作
- **Windows 2000/XP, Solaris 7/8, Linux, MacOS X**などで稼動中
- 現在 α 版の段階

安全プロトコルの概要

- 改竄防止と経路認証
  - 改竄が行われていないことを確認できる
  - 送信者と中継サーバを確認できる
  - 送信を否認できない
- 信頼度の判断材料
  - **E.g.,** 検査サーバが中継したメールは信頼する
- 匿名性での悪事も防ぎたい



安全プロトコルの実現

- 送信者と中継サーバが電子署名
  - 中継のたびに署名を追加
- 付加情報はヘッダーに格納
  - 送信者，送信先，時刻印，次のサーバ，署名など

X-Anz: fm=etsuya@is.titech.ac.jp,to=takuo@cs.titech.ac.jp:..

X-Anz-Sig: sn=0,si=anzenmail.is.titech.ac.jp,di=cs.titech.ac.jp,
ts=1026375710449,sa=SHA1withRSA,sg=xLsXLKk+T1P
l7DxacE1DgWZAQffJ07wQ9sn8luUQZvqlAFOVBD9CP
wZxOP5uJ6nHtQ4y74loMpMokhgfzjdWzafEQzAUMwD
9PBLQK9hlL2v+6xpFdca9pht7OSsLsTzb0BVihZmX666
F93kadO95ceLonKJDEJKAyOGUZ/ll5Eo=

安全プロトコルの検証

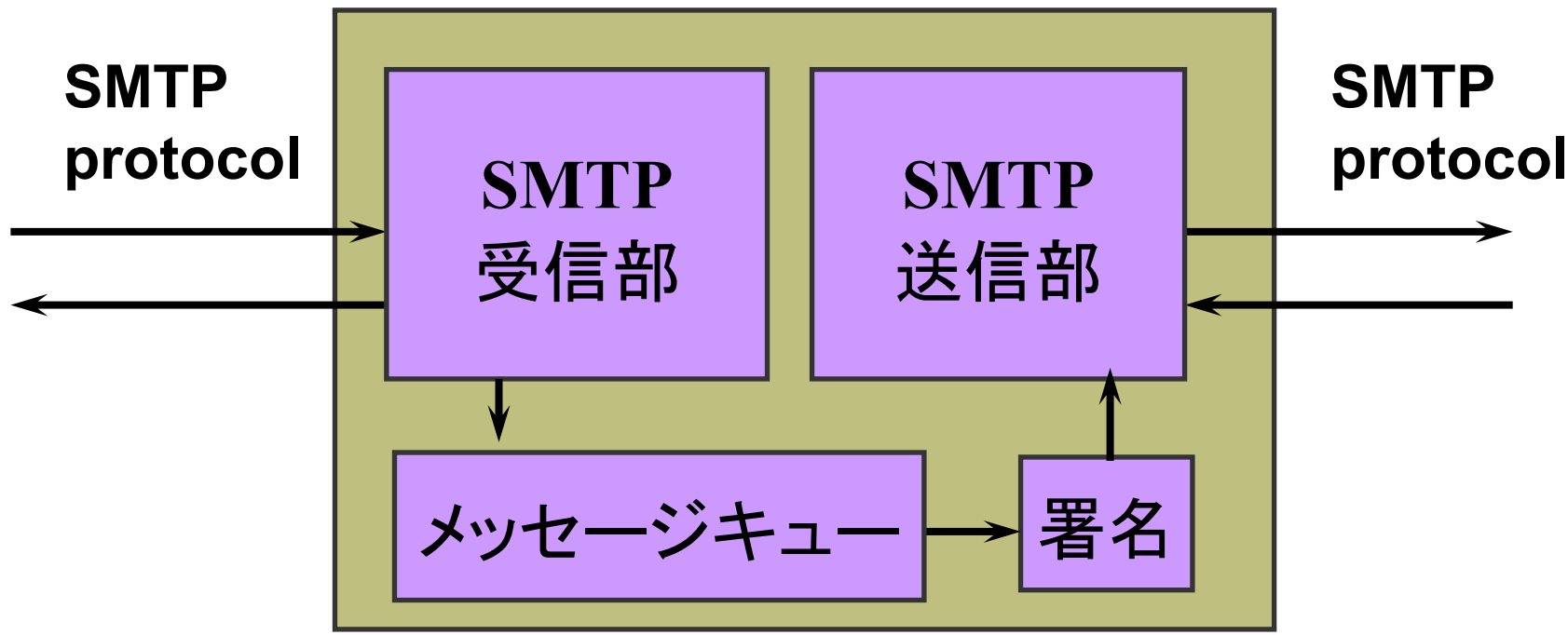
- ほぼ次のような性質を理論的に検証
  - 最終受信者は，メッセージ m を受信した時に，以下を確信できる
    - 送信者が最初のサーバに m を送った
    - 経路上の各サーバ（最後のものは除く）が次のサーバへ m を転送した
- 正確には，もっと複雑な性質を証明している

安全プロトコルと相互運用性

- 配送経路上にレガシー SMTP サーバが存在する場合
  - メールの送受信は可能
  - 経路認証は部分的にしか機能しない
  - レガシーサーバが勝手にメールのボディを変更した場合，改竄とみなされる
- 暗号化，受信否認の防止などについて
  - 他のメッセージフォーマットやプロトコルと組み合わせる
  - SMTP 準拠なら組み合わせ可能

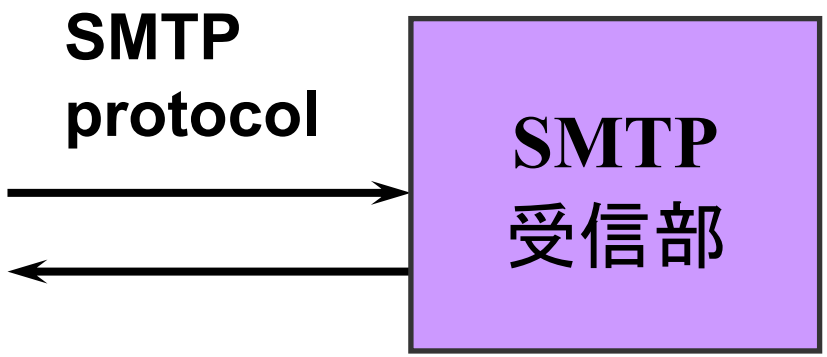
安全サーバの構成

- SMTPの送受信部，メッセージキュー管理部などから構成される，



安全サーバの検証 (1/3)

- **SMTP 受信部の正当性を形式的に検証**
  - プロトコル仕様を満たさないリクエストを拒否する
  - 致命的エラー(**fatal error**)が生じない限り，プロトコル仕様を満たすリクエストを受理する
  - **Ack** を返す前に，メッセージをストレージにセーブする



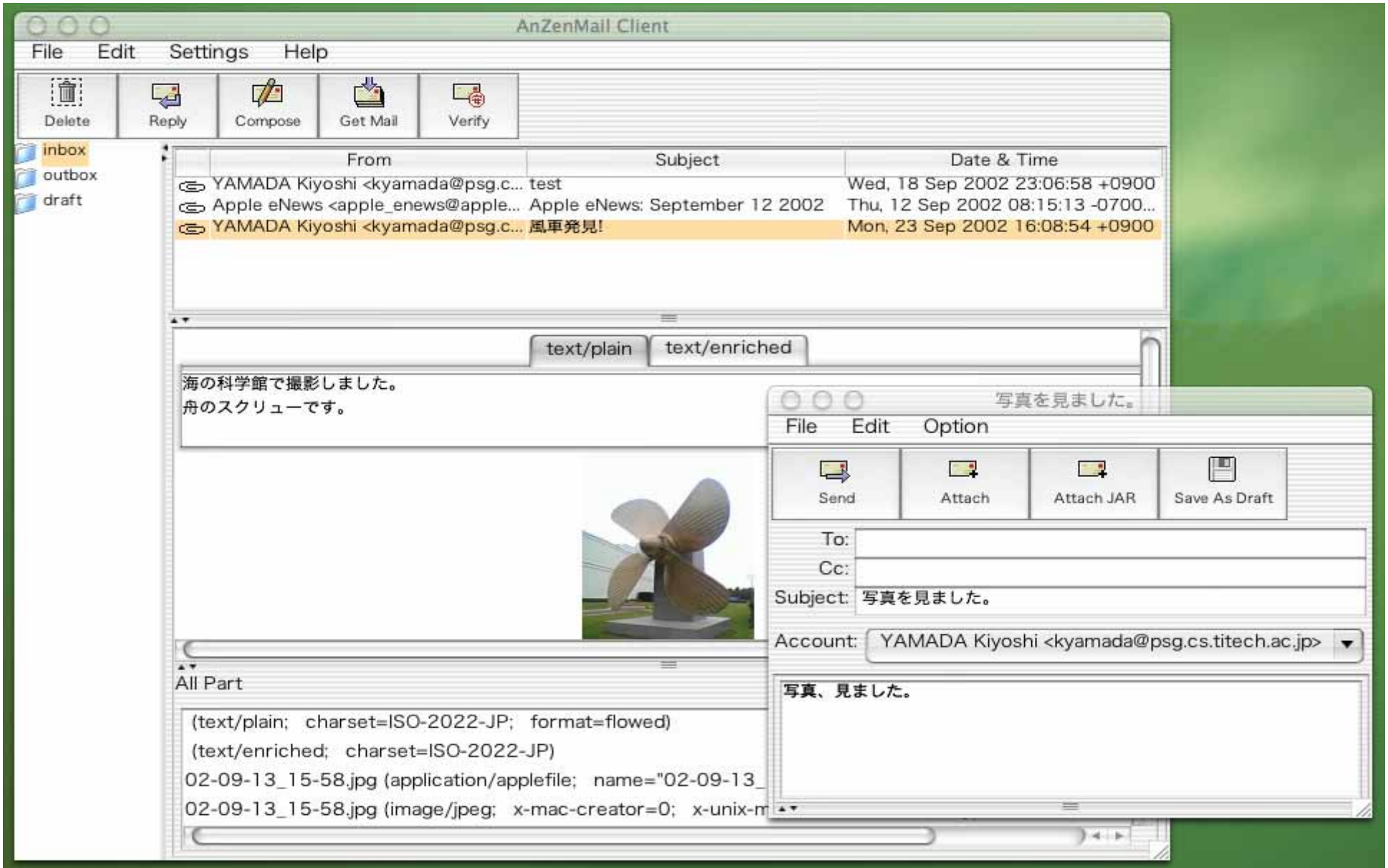
安全サーバの検証 (2/3)

- 証明支援系として **Coq** を利用
  - **Java** のメソッドを **Coq** の関数へ変換してから検証
  - オリジナルのプログラムからバグを数個発見
  - 検証の規模とコスト
    - 対象は約**600**行の**Java**ソースコード
    - 仕様のサイズは約**300**行
    - 証明のサイズは約**5,000**行
    - 要した労力は，約**100**人時＋**2CPU**分

安全サーバの検証 (3/3)

- 検証を可能とした条件
  - 検証の対象が比較的小さかった
  - サーバの記述は，オブジェクト指向的でなく，型安全な命令型
  - スレッド間の干渉が実質的にない

安全メーラ (1/2)

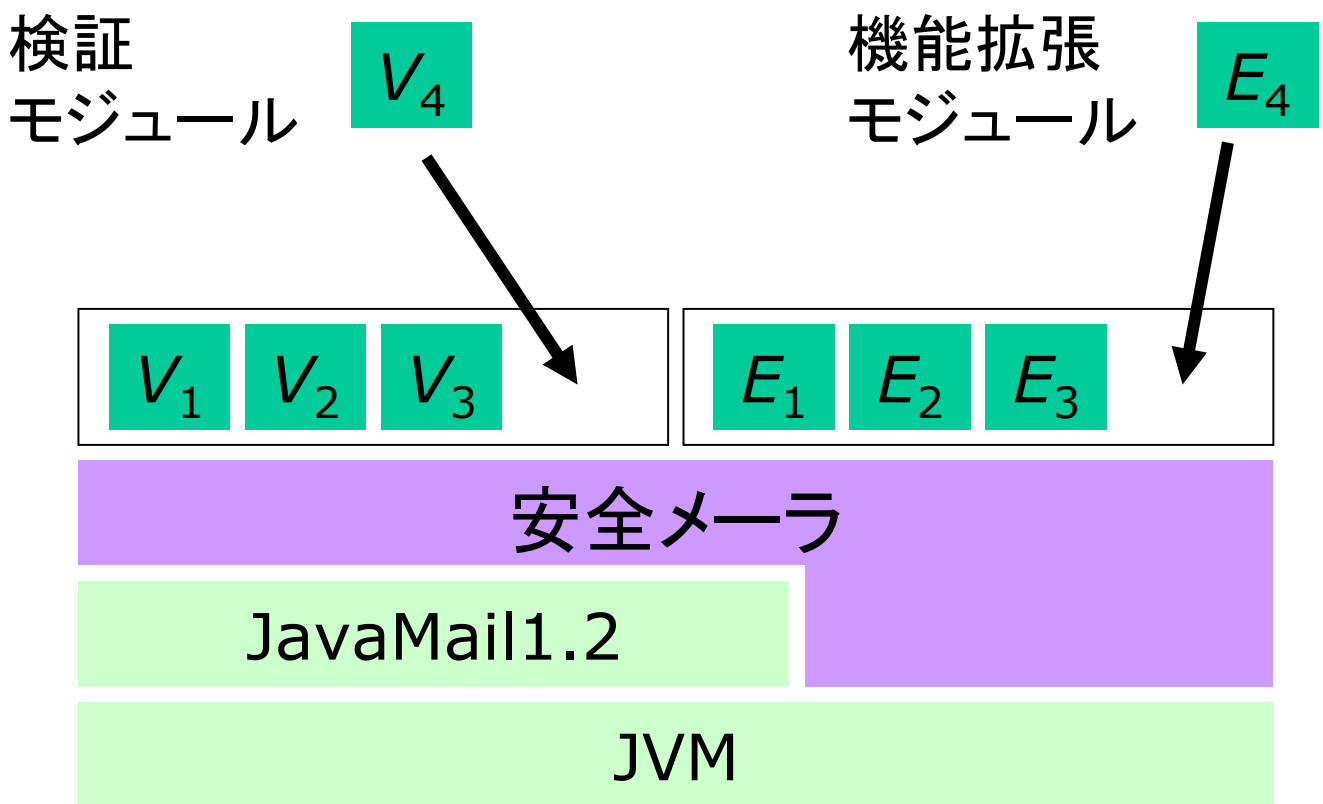

AnZenMail Client
File Edit Settings Help
Delete
Reply
Compose
Get Mail
Verify
inbox
outbox
draft
From
Subject
Date & Time
YAMADA Kiyoshi <kyamada@psg.c... test
Wed, 18 Sep 2002 23:06:58 +0900
Apple eNews <apple_enews@apple... Apple eNews: September 12 2002
Thu, 12 Sep 2002 08:15:13 -0700...
YAMADA Kiyoshi <kyamada@psg.c... 風車発見!
Mon, 23 Sep 2002 16:08:54 +0900
text/plain
text/enriched
海の科学館で撮影しました。
舟のスクリューです。
All Part
(text/plain; charset=ISO-2022-JP; format=flowed)
(text/enriched; charset=ISO-2022-JP)
02-09-13_15-58.jpg (application/applefile; name="02-09-13_
02-09-13_15-58.jpg (image/jpeg; x-mac-creator=0; x-unix-m
写真を見ました。
File Edit Option
Send
Attach
Attach JAR
Save As Draft
To:
Cc:
Subject: 写真を見ました。
Account: YAMADA Kiyoshi <kyamada@psg.cs.titech.ac.jp>
写真、見ました。

安全メーラ (2/2)

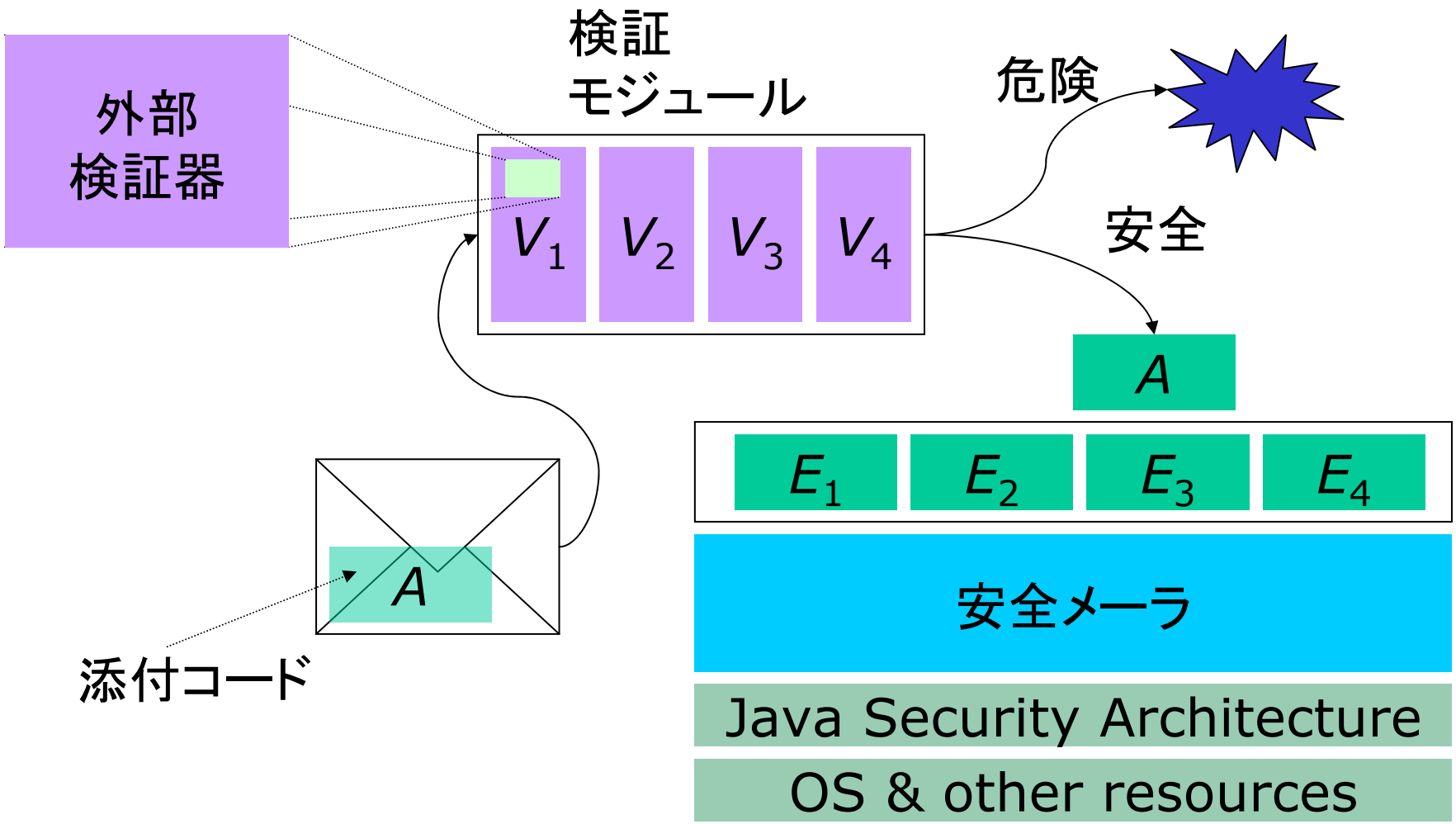
- 見かけは普通のメーラに近い
  - フォルダの一覧
  - 選択されたフォルダ内のメール一覧
  - 選択されたメールの内容表示
- **MIME** タイプに応じて，検証系と表示・実行系を選択
  - 検査系や表示・実行系はプラグイン可能

安全メーラの構成 (1/2)

- 検証系と実行系を追加できる



安全メーラの構成 (2/2)


外部
検証器
検証
モジュール
危険
安全
V_1
V_2
V_3
V_4
A
E_1
E_2
E_3
E_4
A
添付コード
安全メーラ
Java Security Architecture
OS & other resources

外部検証＆実行モジュール

- 署名の検証系
  - Jar ファイルの署名を検査
- **SoftwarePot**
  - **Linux/x86, Solaris** のバイナリをサンドボックス内で実行
- 資源使用解析
  - 資源の利用順序に依存する性質を静的に検査
- アンチウイルスエンジン

まとめ

- 安全な配送プロトコルの設計と検証
  - 改竄と送信・転送の否認を防止
- サーバの構築と検証
  - シンプルな設計
  - コア部の検証
- クライアントの設計と構築
  - 本体の検証は当面行わない
  - 添付ファイルの安全な実行をめざす

各グループの主要成果

A01: 理論的解析・検証

1. Java言語のセキュリティ問題をモデル化、
   新たな欠陥を発見・修正（Sun社に報告）
2. プログラムの計算資源アクセス（順序や回数）の安全性を、型システムにより実行前検査する手法を考案

A02： 言語・記述系

1. 安全なC言語処理系の設計と実現
2. コード挿入により、クライアントの記述したセキュリティーポリシーを強制する機構

A03：OS・インフラ系

1. クラッカーに乗っ取られても、他者に
   害を及ぼさないサーバを実現する機構
2. 資源の過剰使用（DoS攻撃）を防止するOS機構
3. 画像ベースの個人認証システム

そのほかの具体的成果

特定領域研究成果報告

14:45 - 15:05　「研究開発のアプローチと成果概要」　米澤明憲（東京大学）

15:05 - 15:35　「連携研究：安全なメールシステムの構築」　柴山悦哉（東京工業大学）

15:35 - 15:55　「情報セキュリティにおける理論研究の役割」　萩谷昌己（東京大学）

15:55 - 16:15　「新しいウイルス防御法」　森　彰　（産業技術総合研究所）

——休憩——

16:30 - 16:50　「安全なソフトウェア流通実行システム SoftwarePot」

加藤和彦（筑波大学）

16:50 - 17:10　「Moving Firewall: A Defense System against DDoS Attacks」

Eric Y. Chen（NTT/東京大学）

17:10 - 17:30　「ユビキタスコンピューティング環境におけるセキュリティとプライバシ」

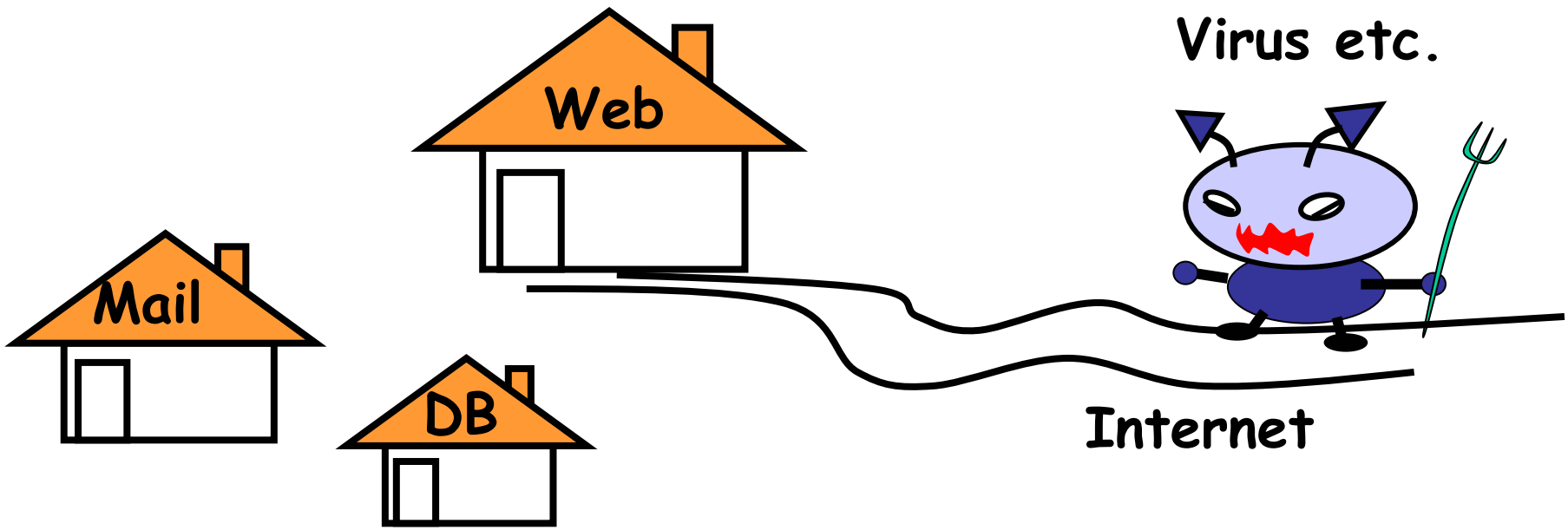
徳田英幸（慶應義塾大学）

17:30 - 17:50　「グリッドサービスのための図形認証とその実証実験」

溝口文雄（東京理科大学）

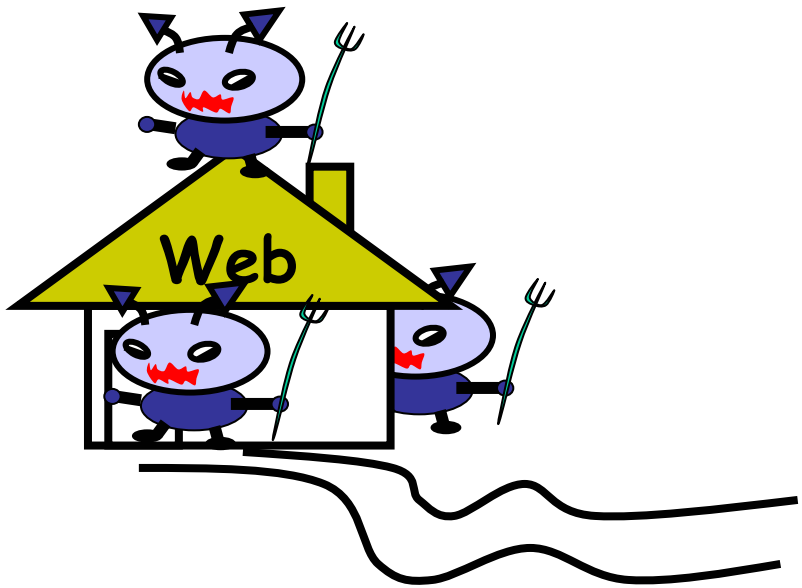
クラッカーに乗っ取られても大丈夫なサーバの構築

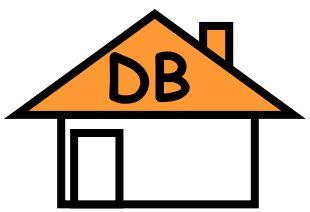
- クラッカーによるサーバへの攻撃



Gee, 乗っ取られた！

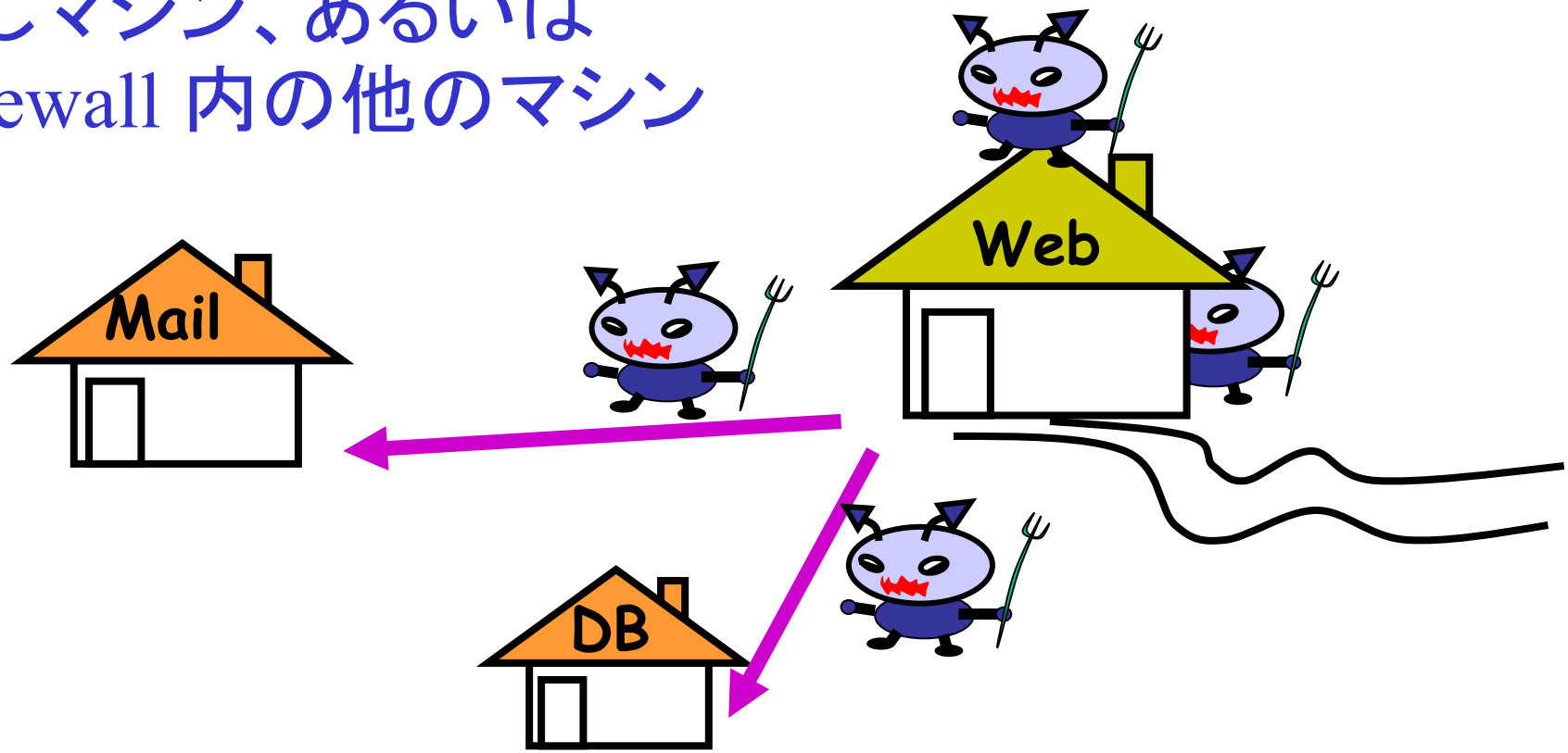
- Buffer Overflow 攻撃、
  あるいは未知の攻撃法





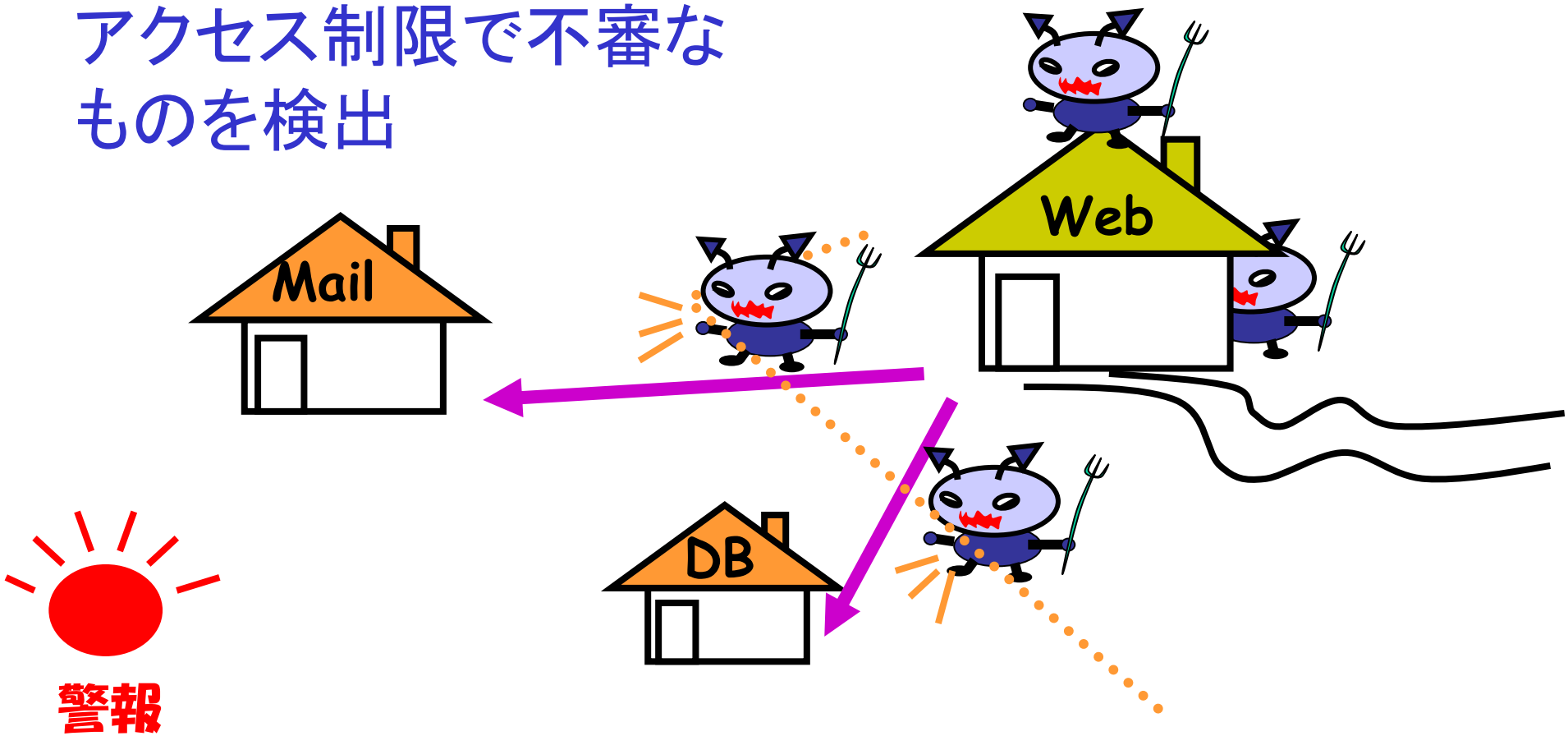
他のサーバへ攻撃が波及

- 同じマシン、あるいは
  Firewall 内の他のマシン



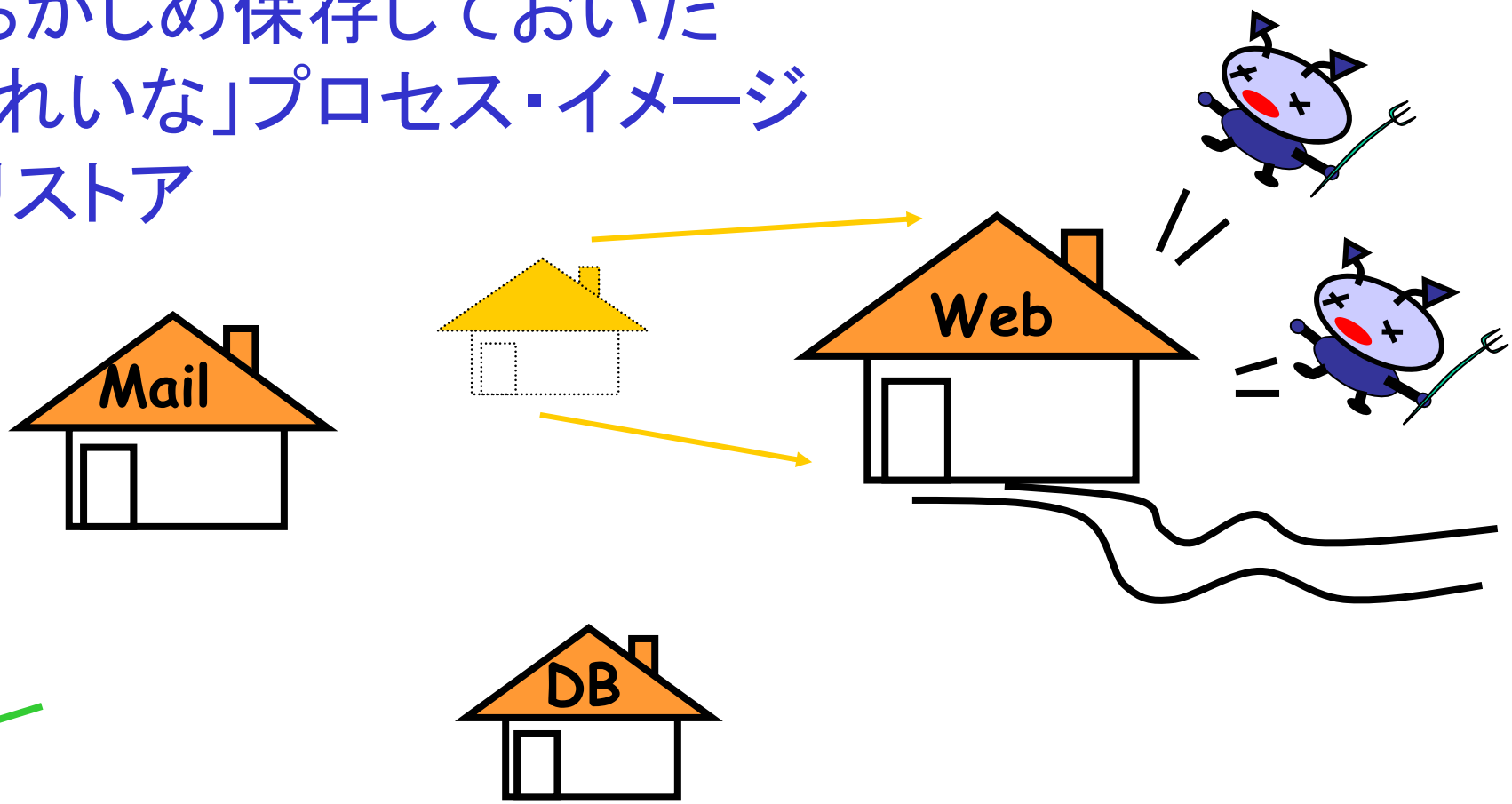
OSが不審なシステムコールを感知

- プロセス単位のきめ細かな
  アクセス制限で不審な
  ものを検出



プロセス・クリーニング [ICDCS01]

- あらかじめ保存しておいた「きれいな」プロセス・イメージをリストア



試作システム

- Linux 2.2.16 カーネルを拡張
- Apache 1.3.12 Web server の動作を確認
  - 全世界の Web server の6割が Apache [Netcraft 調べ]
- 実行時オーバヘッドは最大 35%
  - 類似システムと比較するとオーバヘッドは半分以下に減少

Fail-safeなC言語処設計と実装: 背景

- インターネットの普及とセキュリティーホール
  - 多くのインターネットのサーバアプリケーションは C 言語で記述されている
    - 多くのプログラマが慣れ親しんでいる
    - 汎用性が高い (様々な分野のプログラムの記述が可能)
  - C 言語は「低水準言語」
    - 「メモリ安全性」が保証されない
    - バグの温床

⇒ **C 言語に「メモリ安全性」を付与しよう**

- もっとも基礎的な実行時安全性の必要要件
- 多数の典型的なセキュリティーホール(バッファオーバーフロー等)をカバーできる

Fail-safeなC言語処設計と実装: 概要

- 基本方針: 実行時タグ付け ＋ 型情報の利用
  （Scheme, ML等先進的言語の理論・技術を応用）
  - 配列範囲逸脱を防ぐ「賢いポインタ」の利用
    - バッファオーバーランなどの発生を防ぐ
  - 実行時のタグ付けと型検査
    - ポインタ型変換などによる不正操作を防ぐ
  - 従来処理系と互換性のある実行 semantics
    - 既存プログラムをそのまま利用できることが重要
  - 型情報を用いた最適化による性能向上
    - 安全性を保つ範囲で型検査などを省力化

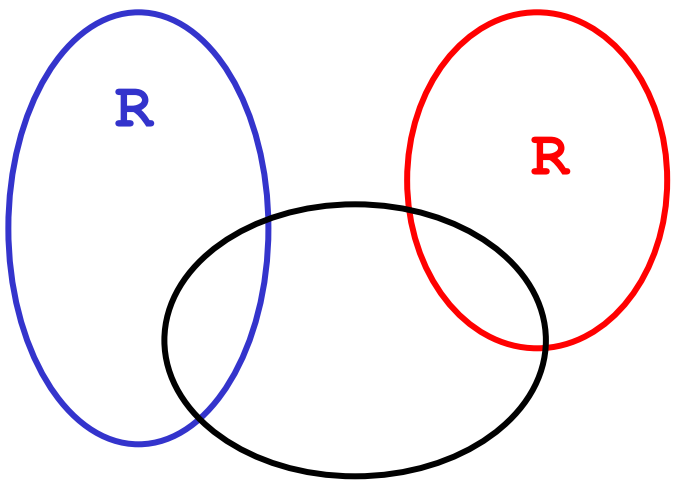
"Fail-safe": 危険な状態に陥ったら即座に停止

⇒ 乗っ取り等の攻撃を防御

Java言語セキュリティの
理論的モデル化
と
その問題点の発見・対策

Java言語のセキュリティ理論モデルに解析

- Java のクラスローダ
  - ローディング・ポリシーのカスタマイズ
  - 名前空間の実現（e.g. アプレットごとの名前空間）
- クラスローダの存在のもとでの型安全性
  - Saraswat の発見した言語設計上のバグ
    - 異なるローダのクラスを混同させ、仮想マシンをクラッシュさせる攻撃が可能



RとRは異なるローダで読み込まれたので、名前は同じだが異なるクラス。

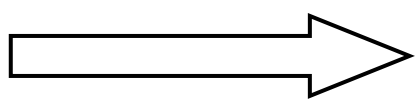
RとRを混同させることが可能。

理論的な解析によるセキュリティ上の新たな問題の発見

- クラスローダの存在のもとでの型安全性
  - Saraswat の発見した言語設計上のバグ
  - Liang & Bracha による解決法
    - **クラスローダ制約 ⇒ Java2**
    - **その正当性は自明ではない。本当に攻撃が防げるのか。**
- 戸沢と萩谷による理論的モデル
  - 理論的モデルによる型安全性の厳密な証明
  - 実装と理論的モデルのずれ ⇒ 新たな攻撃の可能性
    - **Sun に報告 ⇒ 実装の改訂**

クラスローダ制約

$L_1 \overset{\mathcal{C}}{\sim} L_2$



拡張された型付け

$v ::_L^+ n$

証明の核心にある概念