

**Japanese Inter-University  
Research Project  
on Software Security  
-- an overview --**

**Akinori Yonezawa  
University of Tokyo**

# Funding Scheme

- **Sponsor: Ministry of Education and Science**
- **Category: Research-in-Aid for Priority Area**
  - “Research in specific area which leads to the improvement and strengthening of the level in the field of basic scientific research, or which has social impact in order to contribute to the development of economy, society and culture of our country in the 21<sup>st</sup> century”.
  - For each area , 0.6M - 2M dollars/year, for 2-6 years
- **Applicants:**  
**university research groups**

# Selection and Evaluation

- **Each year, ~15 proposals out of ~120 in all science and technology fields. (excl. medical)**
- **Major fields are:**
  - material sciences, physics/chemistry, energy/environment
- **First filtering review based, final selection on oral presentation.**
- **Selected projects have interim and final evaluation**
- **Selection/Evaluation committee members:**
  - dominated by “major” fields’ representatives
  - chaired by a Nobel prize laureate
  - so CS is treated as “poor relative”?

# Our Software Security Project

- Proposal: submitted Dec. 1999, started Sept 2000, ends March 2003.
- Budget: \$1.2M per year
- Organisation: 9 University research groups ( ~35 researchers spread across 5 Japanese universities)  
and one advisory committee

# Subject of Research

- ✓ Enhancing “security of software systems” in {web,mail,computing etc.} servers
- ✓ Excluding cryptography research

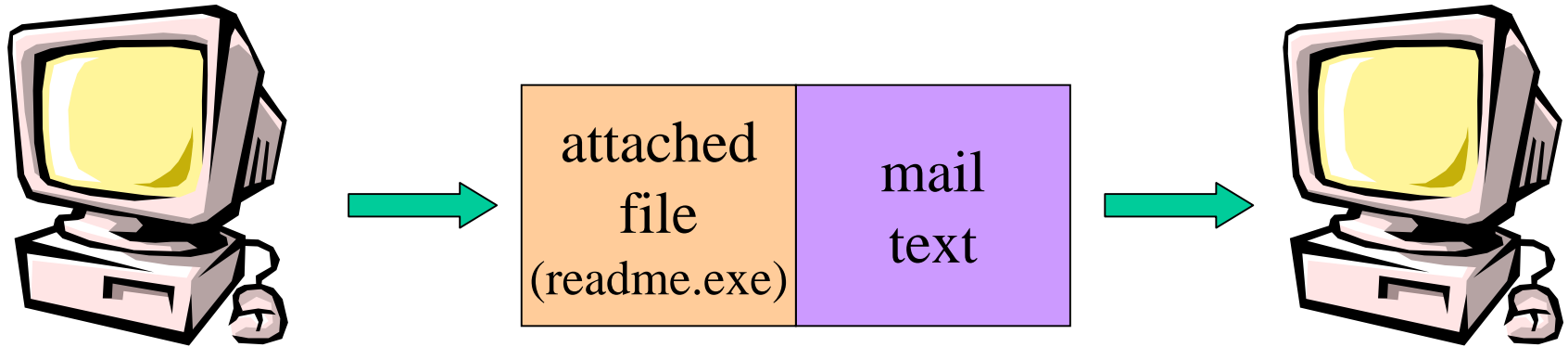
# Types of Security Threats

- **Malicious or erroneous programs**
  - destroy computer software systems .
  - leak information.
- **Computer viruses**
  - intrude computer systems through software security holes.
- **Through “impersonalisation,”**
  - computer systems are used in unauthorized ways
- **Malfunction of software systems in “unexpected situations” (to accommodate non-security experts).**

# **Our Approach in laymen's terms**

# An Intuitive Example

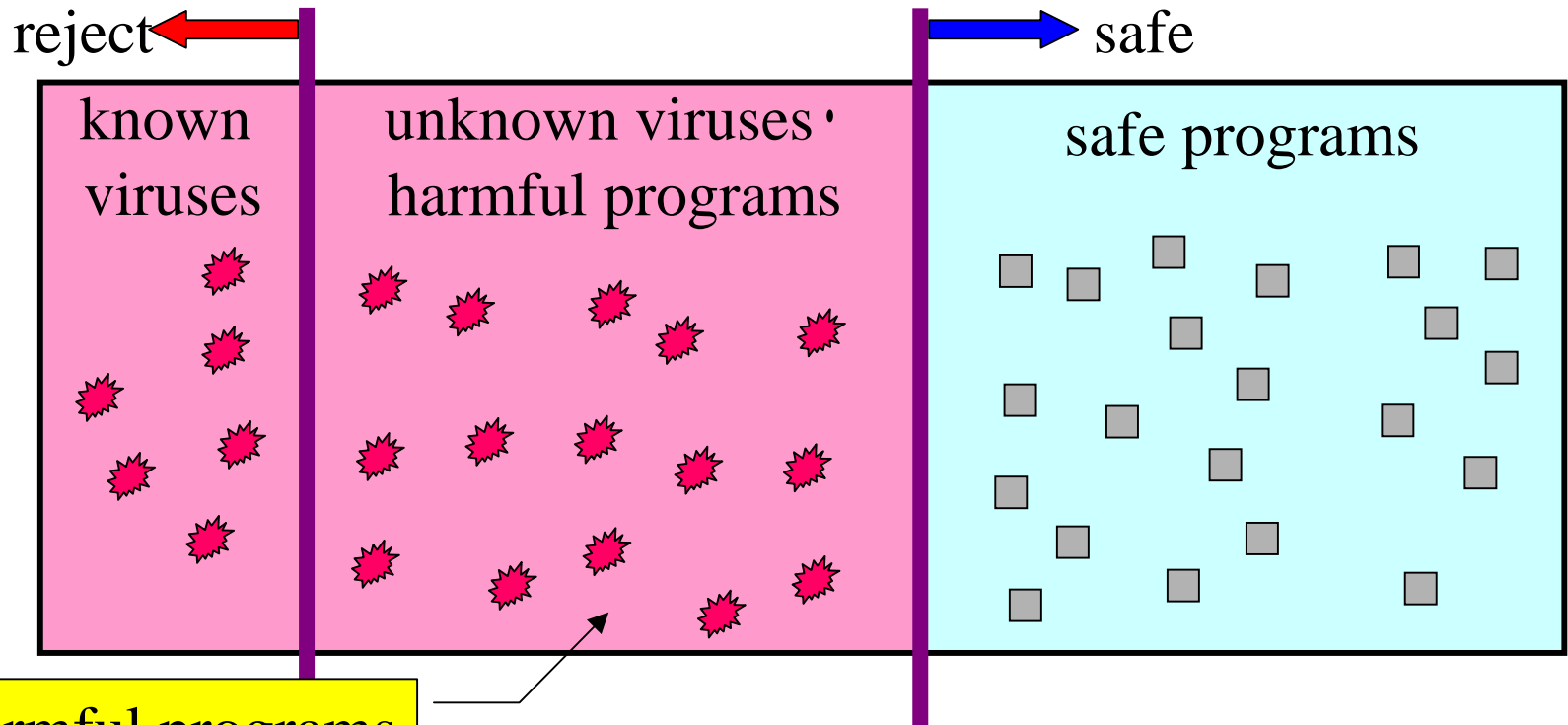
- virus attached to mail message -



- Mail box is always open to outside world
- Simple popular mail systems not designed for malicious files to be attached to mail text.

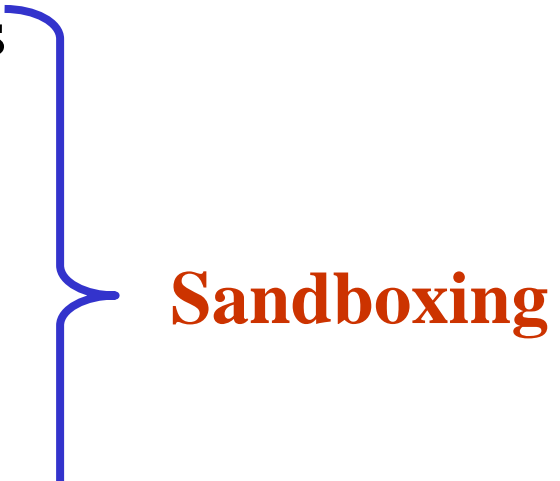
# Previous safety measures

- Don't open attached files
- Use of commercial anti-virus software

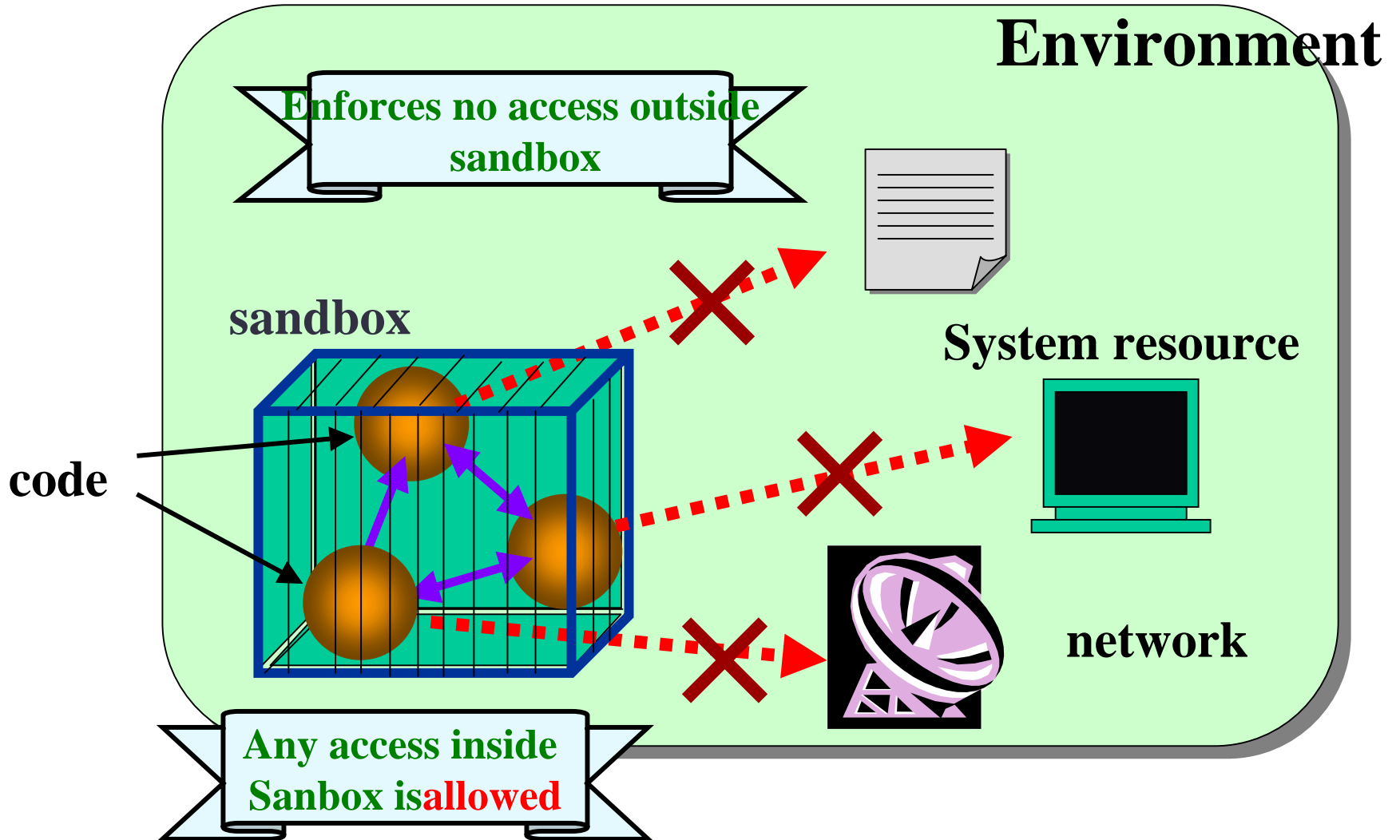


Harmful programs  
can be executed

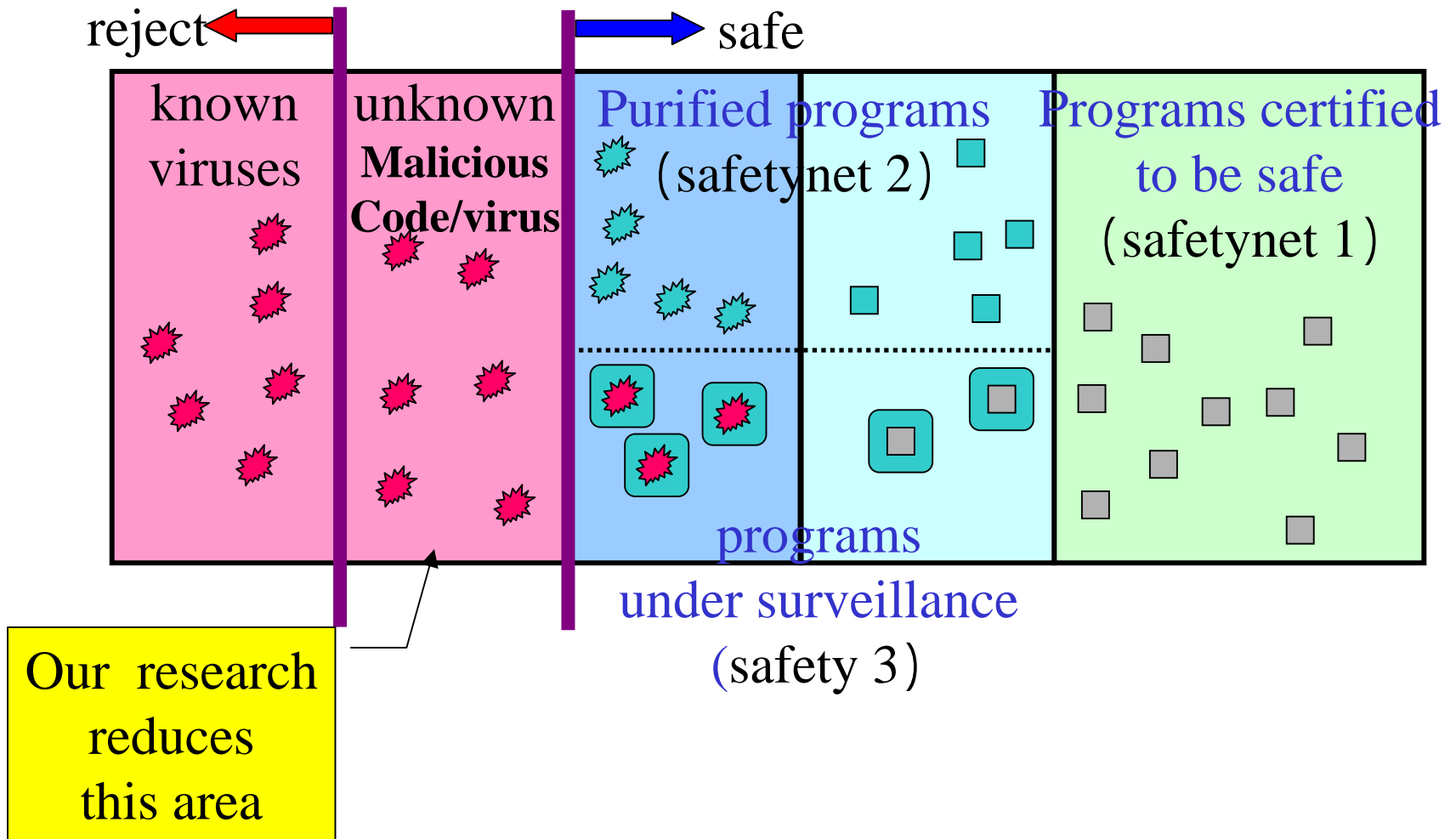
# Our safety measures

- 1 . Checking attached code with theory-based analysis methods. (e.g, static type checking, flow analysis, ...)
  - 2 . Transforming potentially dangerous portions of program into harmless ones  
(by e.g., code insertion,...)
  - 3 . Executing code under surveillance OS or virtual machines
- 
- Sandboxing**

# Sandboxing



# Our safety measures

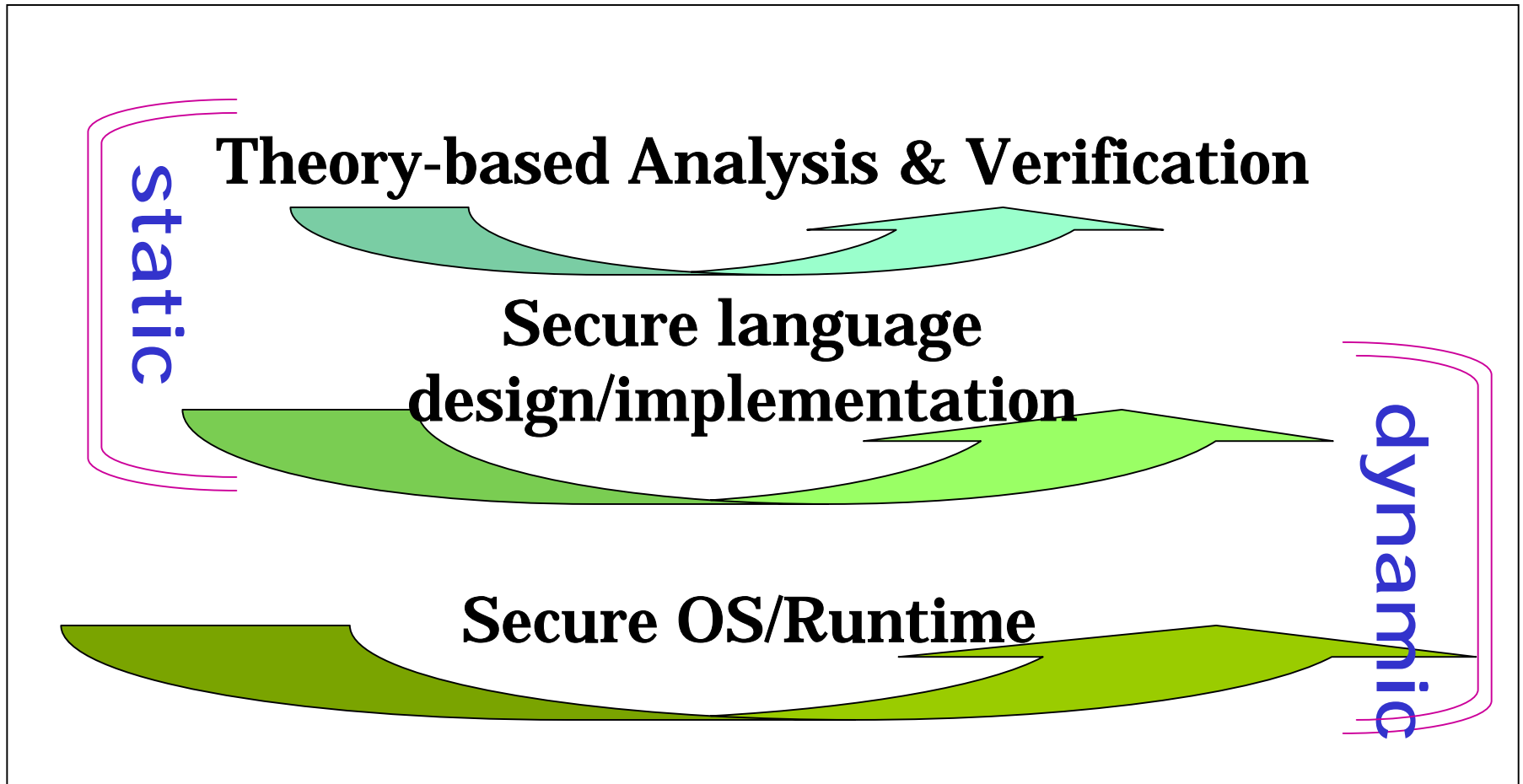


# Three Layers of Safety Nets

- **Theoretical Analysis and Verification on source code and protocols**
  - Proving safety of code at design/construction time
  - Detecting (potentially) dangerous parts of code
- **Programming languages and other description systems**
  - Developing efficient language systems, yet easy to demonstration safety
- **Operating systems and runtime systems**
  - Detecting and suppressing threats not caught by the other two layers

# Our Security Strategy

-- three-layered safety nets --



# Research Groups and Members

A 0 1 (Theoretical analysis and Verification) :

**3 groups represented by**

N. Yonezaki, K. Futatsugi, N. Kobayash,  
M. Hagiya

A 0 2 (Language & Description Systems) :

**3 groups represented by**

A. Yonezawa, E. Shibayama, T. Watanabe

A 0 3 (OS & Infra-structures) :

**3 groups represented by**

H. Tokuda, K. Kato, F. Mizoguchi

# Project Requirements

- **Individual research results**
  - Usual academic contributions
- **Joint (inter-group) research contributions**
  - contributions visible outside CS research community and academia

# Individual Research Results

# Theory-based Analysis & Verification

1. Name-space collision of Java class loaders is formalized, new problems are found and fixed.  
(Sun acknowledged.)
2. Type system for counts and order of resource access.
3. Formal framework for dynamically creating and changing access rights of mobile code .
- 4 . Formal framework for proving confidentiality properties of processes exchanging encrypted messages.
5. Proving some properties of security protocol in BAN logic

# Languages & Descriptive Systems

1. Design/Implementation of a Fail-Safe ANSI-C compiler
2. SSM (Secure Shared Memory):  
dynamic checking scheme for illegal access of mobile / extension code
3. Dynamically enforcing security policies of client code by code insertion

# Operating Systems and Other Infra-Structures

1. Operating system mechanisms that minimize damages of server hijacking (compromising)
2. Operating system mechanisms minimizing damages of DOS attack
3. Implementing campus-wide platform for secure mobile agents
4. Picture/image based personal authentications system (patented in US)

# Joint Inter-Group Research Contribution

- **Constructing a Secure Mail System based on individual skills and research results**
  - at time of proposal submission & review,  
I-Love-You-Mail,
  - just before interim-evaluation,  
Code-Red virus  
Nimda,...

# Design Principles

- **Design minimizing security hole occurrences**
  - accompanied with verification,
  - Use of memory safe languages
- **Even when intruded through security holes, damages are contained**
  - Modules should be given minimum rights, not stronger than necessary
  - Use of fine protection domains
- **Design easy for fixing security holes**

**(Design report available in Japanese)**

# Design Principles

## 1. Separation of core and extension

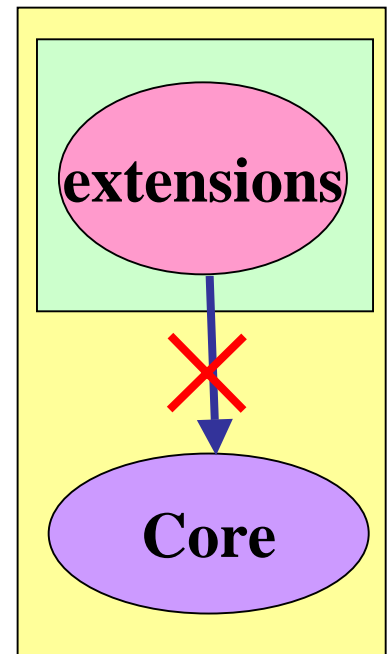
Core: basic mechanisms of sending, transmitting, and receiving

Extension: processing attached files, mailing list, history/log of sending/receiving, etc.

## 2. Theory-based verification on security of the basic structure of the core

## 3. New extensions do not affect the core

## 4. Fine-grained fire-walls embedded in the software



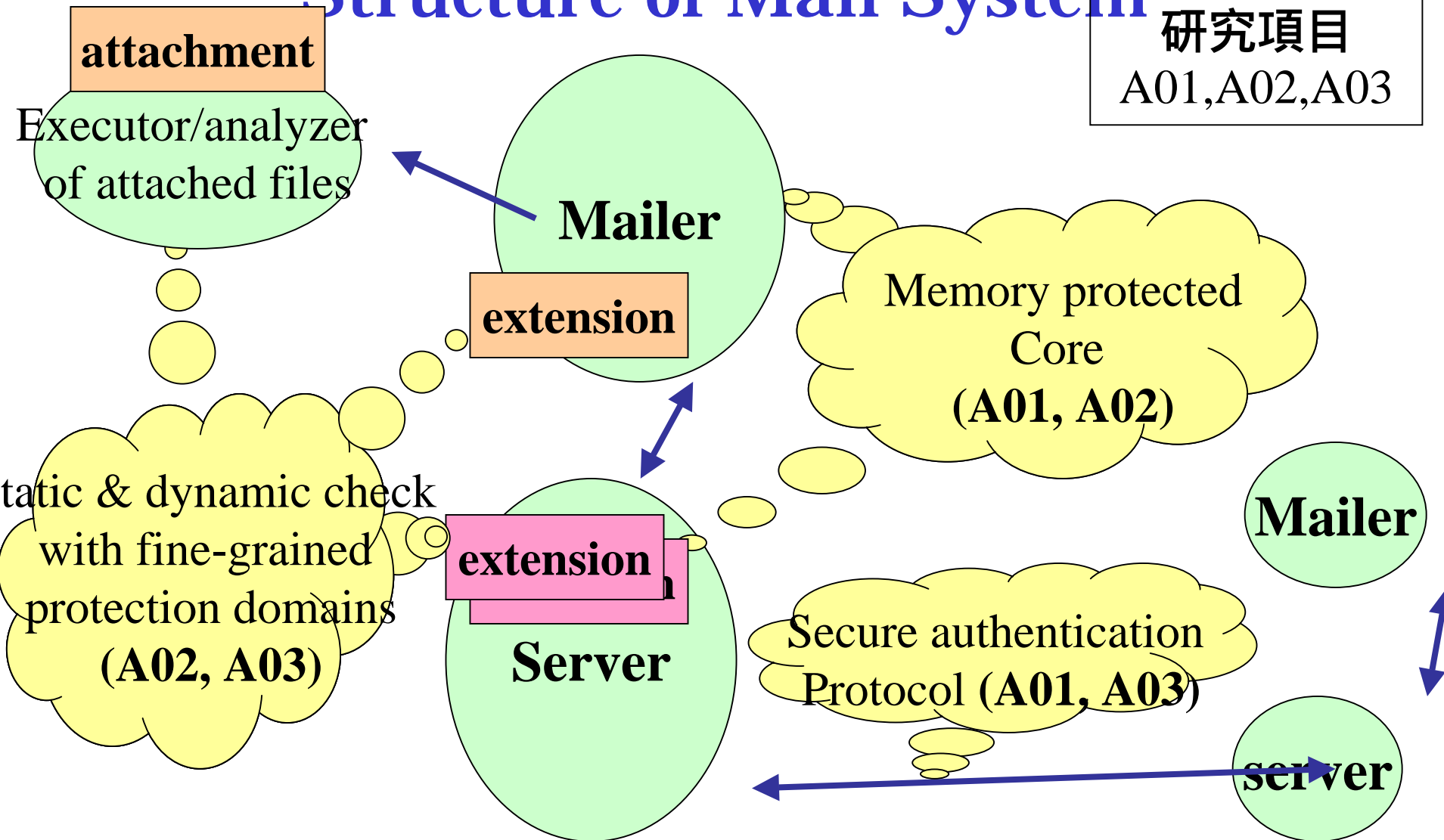
# Security Features of Mail System

- **Powerful analysis and detection on attached files**
  - Conservative analysis and rewriting of attached files and extended modules
- **Powerful Authentication**
  - Possible to rejecting anonymous and spoofed mail message
  - Adopting security policies according to sender's level of trust

# Structure of Mail System

研究項目

A01,A02,A03



# Three-layered Safety Nets for Mail System

- **Verification on source code of Core at design time**
- **Static Check on extension modules at loading time**
- **Dynamic check on loaded modules**

# Overseas Invitees of Our Previous Workshops

- **Jan Vitek, Cedric Fournet,  
Andrew Myers, Dan Wallach**
- **Grit Denker, Drew Dean,  
Giovanni Vigna**
- **George Necula**
- **Doug Tygar, Adrian Perring, D. Song**

# *Implementing a Fail-safe ANSI-C Compiler*

**Y. Oiwa, E. Sumii, T. Sekiguchi**

**A. Yonezawa**

**University of Tokyo**

**October 2001**

# An Approach to Language-level Safety Net

- **Make a safe implementation  
of the full ANSI-C Language**
  - More acceptable approach
    - Existing applications run without modification
    - Current programming style unchanged
  - Is it possible?
    - Can flexible, complex pointer operations be encoded safely and efficiently?

# Fail-safe C Compiler Project: The Goal

- **A fail-safe C language implementation**
  - Full compatibility with ANSI-C
  - Type-aware memory protection
    - Prevent pointer forged from integer
  - Completely safe memory management
    - Prevent access to an already free'd object
  - Support for non-ANSI C features as possible
  - Minimum performance penalty

# Safety and C language (1)

- **Why a C program is unsafe?**
  - Optimistic assumptions on values
    - Pointer access assumes correct pointer/offset
    - Return from function assumes stack soundness
  - “Unspecified behavior” break the assumptions
    - No array boundary check
    - Unchecked pointer type casting

# Safety and C language (2)

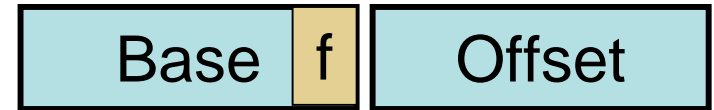
- **To make C language fail-safe:**
  - Exclude optimism on values
    - Distinguish a false pointer from a normal one, and forbid it memory access
  - Exclude disastrous operation son error
    - Array boundary check -- prevent buffer overrun
    - Pointer type checking
      - Ensure atomic values (integer/double) are marked as false pointer if they written to pointer array

# Fail-safe C: Implementation

- **Tagging every pointer and integer**
  - Whether the value may be a valid pointer
  - Which region the pointer points to
- **Annotate every memory region with**
  - The number of elements in the region
  - The type of values stored in it
- **(Preserve backward-compatible behavior**
  - Hide internal representation from a program)

# Pointer representation

- **2 words**



- $\langle$ Base, offset $\rangle$  pair

- Base always points to the region top (or 0 if NULL)

- 1-bit flag for cast(ed) pointers ( $f$ )

- $f = 0$ : The referred object has correct type

- For fast access with not-cast(ed) pointer

- $f = 1$ : Access needs type checking

- It was cast(ed) to another type (e.g. void \*)

# Integer representation



- **2 words**

- ANSI C requires:

- There is an integer type which holds void\* value
- Pointers cast(ed) to that type may be re-cast(ed) to void \*

The runtime must distinguish such an integer holding a valid pointer with other integers

- Future optimization: Use 1-word representation if it is possible