CertiKOS: From Hacker-Resistant OS to Certified Heterogeneous Systems

Zhong Shao
Yale University
November 14, 2022

Acknowledgement: Ronghui Gu, Newman Wu, Hao Chen, Jeung Kim, Jeremie Koenig, Vithgen Sjoberg, Mengqi Liu, Man-Ki Yoon, Jung-Eun Kim, Lionel Reg, Quentin Carbonneaux, Unsung Lee, J. Yong Shin, David Costanza, Yahya Ramananrind, Hornain Varette, Shi-Chun Weng, Zifeng Zeng, Zhuoqi Zhang, Liang Gu, Jan Hofmann, and Joshua Lockerman. This research is supported in part by DARPA CRASH and HACMS programs and NSF SaTC and Expeditions in Computing programs.

The Future of CS: Heterogeneous Systems

- **Hardware Components**
  - Multicore CPU, MMU/OOMMU, Cache, FPGA, GPU, TPU, ASIC, Domain-Specific HW, ...
  - Cyber-physical systems, smart cars, robotics, IoTs, smart cities, blockchains, clouds, ...
- **Human and Social Components**
  - Passengers, pedestrians, insurance companies, lawyers, policy-makers, ...
  - Banks, consumers, currencies, smart contracts, crypto coins, DAO, federal reserve, ...
- **OS Components**
  - Processes, schedulers, containers, device drivers, virtual machines, file systems, sockets, databases, logs, atomic objects, transactions, network stacks, security protocols, ...
- **SW & PL Components**
  - Data structures; objects; transactions; modules; threads; interrupt handlers; exception; communication channels; concurrent & distributed objects; containers; micro-services; ...
  - Language specs, compilers, interpreters, JIT, binary translator; parsers, serializer / pretty printer, ...
- **Challenge: how to establish strong trust & accountability properties?**
  - Safety, security (isolation), resource efficiency, availability, accountability, extensibility, ....

PL Research: The Charm

- Uncover the **essence** (i.e., semantic abstraction) of various (SW & HW) systems, or **systems of systems**
- Use these new theories to **bridge** different areas of specialty (e.g., via new lang., compilers, tools) and then **build** better & more secure/reliable systems

---

Rich Logical Foundations:
- category theory
- universal algebra & co-algebra
- inductive vs. coinductive types
- classical vs. intuitionistic vs. linear logics
- monad vs. comonad
- equality (HTT) vs simulation
- Curry-Howard correspondence
- lambda & process calculi
- denotational vs operational semantics

New PLs, DSLs, and Compilers:
- static vs dynamic typing vs. “no typing”
- explicit resource manager vs. GC
- effects vs. encapsulation
- OO vs. function programming
- New multiparadigm languages: Java Scala, Swift, Rust, Go, C++, C#
  - Concurrent vs. parallel vs distributed programming
  - proof assistant languages
  - information flow control & security
  - certified software & compilers

PL Meets OS: An Ideal Marriage Yet to Happen?

- PL is to uncover the laws of abstraction in the cyber world
- PL is to use abstraction to reduce complexities
- PL depends on the underlying OS for sys lib.
- Many PL issues are easily resolved in OS
- OS is to build layers of abstraction (i.e., VMs) for the cyber world
- OS is full of complexities
- OS is to manage, multiplex, and virtualize resources
- OS really needs PL help to provide safety and security guarantee
The CertiKOS / DeepSpec Project

**Killer-app:** high-assurance “heterogeneous” systems of systems!

**Conjecture:** today’s PLs fail because they ignored OS, and today’s OSes fail because they get little help from PLs

**New Insights:**
- deepspec & certified abstraction layers;
- a unifying framework for composing heterogeneous components (via game semantics + linear logic connectives)

**Opportunities:**
- New certified system software stacks (CertiKOS ++)
- New certifying programming languages (DeepSEA vs. C & Asm)
- New certified programming tools
- New certified modeling & arch. description lang. (DeepSEA)
- We verify all interesting properties (correctness, safety, security, availability, …)

Hacker-Resistant OS: Why?
Hacker-Resistant OS: Why?

CertiKOS Problem Definition

• What is a certified OS?
  – an OS binary implements its specification?
  – what should its specification be like?

• What properties do we want to prove?
  – safety & partial correctness properties
  – total functional correctness
  – security properties (isolation, confidentiality, integrity, availability)
  – resource usage properties (stack overflow, real time properties)
  – race-freedom, atomicity, and linearizability
  – liveness properties (deadlock-freedom, starvation freedom)

• How to cut down the cost of verification?

Motivation

Formal Verification

"Formal verification is the only way to guarantee that a system is free of programming errors."

— NSF SFM Report[2016]

"Formal methods are the only reliable way to achieve security and privacy in computer systems."

Challenges: huge proof efforts

seL4 [SOSP’09]

Proof 11 py

C 7.5k LOC

Asm 500 LOC unverified

C 1.3k LOC unverified
Challenges: Compositionality

Abstraction Gap
Challenges: Compositionality

A Complex System

Complete Verification

Compiler

Asm

Challenges: Concurrency

fine-grained lock

I/O concurrency

multi-thread

multiprocessor

Challenges: Concurrency

CPU i

Complete Verification

CPU j
Certified Abstraction Layers

CertiKOS

aim to solve all these challenges

verify existing systems

build the next generation heterogeneous systems
designed to be reliable and secure

CertiKOS

CPU i

CPU j

fine-grained lock

untangle

verify existing systems

build certified heterogeneous systems
Contribution

Certified Abstraction Layers

R1
M1
L1
R0
M0
L0

R1
M1
L1
R0
M0
L0

Contribution

Certified Abstraction Layers

mCertiKOS [POPL’15]
certified sequential OS kernels
3k C&Asm, 1 py

Interrupt [PLDI’16a] 0.5 py

Security [PLDI’16b] 0.5 py

mC2 [OSDI’16] [PLDI’18]
the first formally certified concurrent
OS kernel with fine-grained locks
6.5k C&Asm, 2 py

CompCertX

Asm
L0

CompCertX

Asm
L0

C
L0

C
L0

mCertiKOS [POPL’15]
certified sequential OS kernels
3k C&Asm, 1 py

Interrupt [PLDI’16a] 0.5 py

Security [PLDI’16b] 0.5 py

mC2 [OSDI’16] [PLDI’18]
the first formally certified concurrent
OS kernel with fine-grained locks
6.5k C&Asm, 2 py

CompCertX

Asm
L0

CompCertX

Asm
L0

C
L0

C
L0
**Certified System Software**

- Functional correctness
- Liveness
- No stack/integer/buffer overflow
- No race condition

---

**mC2**

- 6.1k LOC (C layers)
- 400 LOC (Asm layers)

**Deployment**

CertiKOS on Landshark, DARPA HACMS
Deployment

CertiKOS on Quadcopter

Case Study

Build a Certified System

Certified Sequential Layer [POPL'15]

Certified Sequential Layer [POPL'15]
Example: Thread Queue

typedef struct tcb {
    state s;
    tcb *prev, *next;
} tcb;
tcb tcbp[1024];

typedef struct tdq {
    tcb *head, *tail;
} tdq;
tdq* td_queue; C


typedef struct tcb {
    state s;
    tcb *prev, *next;
} tcb;
tcb tcbp[1024];
tcb* td_queue;

typedef struct tdq {
    tcb *head, *tail;
} tdq;
tdq* td_queue;

tcb* dequeue(tdq* q) {
    tcb *head, *next;
tcb *i = null;
    if (!q) return i;
    head = q -> head;
    if (!head) return i;
i = head;
    next = i -> next;
    if (!next) {
        q -> head = null;
        q -> tail = null;
    } else {
        next -> prev = null;
        q -> head = next;
    }
    return i;
}
Example: Thread Queue

specification

```
<table>
<thead>
<tr>
<th>tcbp(0) tcbp(1) tcbp(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0 s1 s2</td>
</tr>
<tr>
<td>-------------------------</td>
</tr>
<tr>
<td>1 :: 0 :: 2 :: nil</td>
</tr>
</tbody>
</table>
```

Function `dequeue(q) :=
match q with
| head :: q' => (q', Some head)
| nil => (nil, None)
end.`

Coq

Example: Thread Queue

specification

```
<table>
<thead>
<tr>
<th>tcbp(0) tcbp(1) tcbp(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0 s1 s2</td>
</tr>
<tr>
<td>-------------------------</td>
</tr>
<tr>
<td>0 :: 2 :: nil</td>
</tr>
</tbody>
</table>
```

Function `dequeue(q) :=
match q with
| head :: q' => (q', Some head)
| nil => (nil, None)
end.`

Coq

Simulation Proof

specification

```
<table>
<thead>
<tr>
<th>L2</th>
</tr>
</thead>
</table>

Program Context
```

Deep Specification [POPL'15]

```
Deep spec L2 captures all we
need to know about M over L1

Any property about M can be
proved using L2 alone

No need to look at M again
```

Deep Speciﬁcation

```
M
```
mCertiKOS

kernel

code

seq machine

mCertiKOS

memory management

seq machine

mCertiKOS

kernel

Trap
PM
TM
MM

seq machine

mCertiKOS

trap
proc
thread
mem

seq machine

mCertiKOS

kernel

Trap
PM
TM
MM

seq machine
mCertikOS

certified sequential kernel

- trap
- proc
- thread
- mem
- seq machine

mCertikOS

certified hypervisor

- trap
- proc
- thread
- mem
- seq machine
mCertiKOS

3k LOC

[POPL'15]

Can boot Linux as a guest

Concurrent Framework [OSDI'16, PLDI'18]

certified sequential kernel

trap

virt

proc

thread

mem

seq machine

multicore machine

Concurrent Framework [OSDI'16, PLDI'18]

certified concurrent layer

contribution

machine lifting

seq machine

CPU-local machine

multicore machine

Concurrent Framework [OSDI'16, PLDI'18]

certified concurrent layer

contribution

machine lifting

spin-lock

seq machine

CPU-local machine

multicore machine
Contribution

Certified Concurrent Abstraction Layers

Case Study

```
struct ticket_lock {
    volatile uint n, t;
};
//Methods provided by L0
extern void acq();
extern void rel();
extern cpu_id();
//M2 module
void acq () {
    uint my_t = FAI_t();
    while(get_n()!=my_t){
        hold();
    }
}
void rel () {
    inc_n();
}
//Methods provided by L1
extern void acq();
extern void rel();
extern cpu_id();
#include<asm.h>
//Client program P
//Thread running on CPU 1
void T1 () { update_x(); }
//Thread running on CPU 2
void T2 () { update_x(); }
```

```
#include<asm.h>
//Methods provided by L1
extern void acq();
extern void rel();
extern cpu_id();
#include<asm.h>
//Client program P
//Thread running on CPU 1
void T1 () { update_x(); }
//Thread running on CPU 2
void T2 () { update_x(); }
```
Case Study

```c
struct ticket_lock {
    volatile uint n, t;
}; //Methods provided by L0
extern void acq();
extern void rel();
extern cpu_id();
//M1 module
void acq () {
    uint my_t = FAI_t();
    while(get_n() != my_t){}
    hold();
}
void rel () {
    inc_n();
}

//Methods provided by L1
extern uint get_n();
extern void inc_n();
extern uint FAI_t();
extern void hold();

//M2 module
int x = 0; //shared variable x
void update_x () { //Client program P
    acq(); x += cpu_id(); rel();
}
```

```
//Methods provided by L2
void update_x() {
    acq(); x += cpu_id(); rel();
}
```

//Client program P
```c
void T1 () { update_x(); }
void T2 () { update_x(); }
```

Strategies and Game Semantics

**strategy** \( \psi_p[i] \)

How will the program \( p \) generate events on behalf of CPU \( i \) at each step regarding the given logical log \( l \)?
void acq () {
    uint my_t = FAI_t();
    while(get_n()!=my_t) {
        hold();
    }
}
Strategies and Game Semantics

void acq () {
    uint my_t = FAI_t();
    while(get_n()!=my_t){};
    hold();
}

void rel () {
    inc_n();
}

M1 module
void acq () {
    uint my_t = FAI_t();
    while(get_n()!my_t){};
    hold();
}
void rel () {
    inc_n();
}

M2 module
int x = 0; //shared variable x
void update_x () {
    acq(); x += cpu_id(); rel();
}

Strategy (L0[i])

Given the current log i, how the module M0 running
over [Lo[i]] will generate events on behalf of CPU i at each
step.

Strategies and Game Semantics
Strategies and Game Semantics

?l, !1.FAI_t, $t

Methods provided by L0
extern uint get_n();
extern void inc_n();
extern uint FAI_t();
extern void hold();
//M1 module
void acq () {
  uint my_t = FAI_t();
  while(get_n()!=my_t){}
  hold();
}
void rel () { inc_n(); }

//M2 module
int x = 0; //shared variable x
void update_x () {
  acq(); x += cpu_id(); rel();
}
//Methods provided by L2
extern void update_x();
//Client program P
//Thread running on CPU 1
void T1 () { update_x(); }
//Thread running on CPU 2
void T2 () { update_x(); }

//Client program P
//Thread running on CPU 1
void T1 () { update_x(); }
//Thread running on CPU 2
void T2 () { update_x(); }

C
Strategies and Game Semantics

//Methods provided by L₀
extern uint get_n();
extern void inc_n();
extern uint FAI_t();
extern void hold();

//M₁ module
void acq () {
    uint my_t = FAI_t();
    while(get_n()! = my_t){};
    hold();
}
void rel () { inc_n(); }

//Methods provided by L₂
extern void update_x();

//Client program P
//Thread running on CPU 1
void T₁ () { update_x(); }
//Thread running on CPU 2
void T₂ () { update_x(); }

//Methods provided by L₀
int x = 0; //shared variable x
void update_x () {
    acq(); x += cpu_id(); rel();
}

//Methods provided by L₂
void acq () {
    uint my_t = FAI_t();
    while(get_n()! = my_t){};
    hold();
}
void rel () { inc_n(); }

//Methods provided by L₀
int x = 0; //shared variable x
void update_x () {
    acq(); x += cpu_id(); rel();
}

//Methods provided by L₂
void acq () {
    uint my_t = FAI_t();
    while(get_n()! = my_t){};
    hold();
}
void rel () { inc_n(); }

//Methods provided by L₀
\[E_{hs} \rightarrow 1 \, 2 \, 2 \, 1 \, 1 \, 2 \, 1 \, 2 \, 1 \, 2 \, 2 \, 2 \, 2 \]

**Logical Flow**

1. **Log 1**
   - \(1, 1.\text{get}_n, \text{Sn}=1\)
   - \(1, 1.\text{FAI}_t, \text{Sn}=1\)
   - \(1, 1.\text{hold}, \text{Sn}=1\)
   - \(1, 1.\text{inc}_n, \text{Sn}=1\)

2. **Log 2**
   - \(1, 2.\text{get}_n, \text{Sn}=1\)
   - \(1, 2.\text{FAI}_t, \text{Sn}=1\)
   - \(1, 2.\text{hold}, \text{Sn}=1\)
   - \(1, 2.\text{inc}_n, \text{Sn}=1\)
Strategy Refinement

\[ R_j \neq i : \text{will release lock within } k \text{ steps} \]
\[ \psi_{acq}^{[i]} \]

Add fuel (f) to prove liveness

\[ R_{hs} : \text{(fairness) each CPU will be rescheduled within } m \text{ steps} \]
\[ \psi_{acq}^{[i]} \]
Strategy Refinement

$R_j \neq i$ : will release lock within $k$ steps

$\psi_{acq}[i]$ : (fairness) each CPU will be rescheduled within $m$ steps

$R_{cpu}$ : $\#CPU = c$ is bounded

mutual exclusion?

Certified Concurrent Abstraction Layer

$R_j \neq i$ : will release lock within $k$ steps

$\psi_{acq}[i]$ : (fairness) each CPU will be rescheduled within $m$ steps

$R_{cpu}$ : $\#CPU = c < 2^{32}$
Certified Concurrent Abstraction Layer

\[
\begin{align*}
(M_{\text{acq}})_{L_{0[i]}} & \leq_{R} \psi_{\text{acq}[i]} \\
R \quad M_{\text{acq}} & \quad L_{0[i]} \\
\psi_{\text{rel}[i]} & \quad \psi_{\text{rel}[i]} \\
R \quad M_{\text{rel}} & \quad L_{0[i]}
\end{align*}
\]

Horizontal Composition

\[
\psi_{\text{acq}[i]} \oplus \psi_{\text{rel}[i]}
\]

Horizontal Composition

\[
\psi_{\text{acq}[i]} \oplus \psi_{\text{rel}[i]}
\]

Certified Concurrent Abstraction Layer

\[
L_{1[i]}
\]

R \quad M_{\text{acq}} \oplus \quad M_{\text{rel}}

L_{0[i]}
void update_x () {
    acq();
    x += cpu_id();
    rel();
}
Parallel Composition

\[ R_{ij} \neq i : \text{will release lock within } k \text{ steps} \]

\[ R_{hs} : \text{(fairness) each CPU will be rescheduled within } m \text{ steps} \]

\[ R_{cpu} : \#CPU = c < 2^{32} \]

Parallel Composition

\[ R_{ij} \neq i : \text{will release lock within } k \text{ steps} \]

\[ R_{hs} : \text{(fairness) each CPU will be rescheduled within } m \text{ steps} \]

\[ R_{cpu} : \#CPU = c < 2^{32} \]

Parallel Composition

\[ R_{hs} : \text{(fairness) each CPU will be rescheduled within } m \text{ steps} \]

\[ R_{cpu} : \#CPU = c < 2^{32} \]

Parallel Composition

\[ R_{hs} : \text{(fairness) each CPU will be rescheduled within } m \text{ steps} \]

\[ R_{cpu} : \#CPU = c < 2^{32} \]

Case Study

\[ R_{hs} : \text{(fairness) each CPU will be rescheduled within } m \text{ steps} \]

\[ R_{cpu} : \#CPU = c < 2^{32} \]
**Case Study**

\[ \begin{align*}
\gamma_1 & \text{, } \gamma_1 \text{.get_n, } \gamma_1 \text{.hold, } \gamma_1 \text{.x+=1, } \gamma_1 \text{.inc_n, } \\
\gamma_2 & \text{, } \gamma_2 \text{.get_n, } \gamma_2 \text{.hold, } \gamma_2 \text{.x+=2, } \gamma_2 \text{.inc_n, }
\end{align*} \]

\[ \mathcal{E}_{hs} : \begin{array}{cccccccc}
1 & 2 & 2 & 1 & 1 & 2 & 1 & 2 \\
1 & 2 & 2 & 1 & 1 & 2 & 1 & 2 \\
1 & 2 & 2 & 1 & 1 & 2 & 1 & 2 \\
1 & 2 & 2 & 1 & 1 & 2 & 1 & 2 \\
1 & 2 & 2 & 1 & 1 & 2 & 1 & 2
\end{array} \]

\[ R \circ R' \]

\[ \mathcal{E}'_{hs} : \begin{array}{cccccccc}
1 & 2 & \ldots & & & & & \\
1 & 2 & \ldots & & & & & \\
1 & 2 & \ldots & & & & & \\
1 & 2 & \ldots & & & & & \\
1 & 2 & \ldots & & & & & \\
\end{array} \]

**Soundness**

\[ R_{hs} : \text{(fairness) each CPU will be rescheduled within } m \text{ steps} \]

\[ R_{cpu} : \#CPU = c < 2^{32} \]

\[ \begin{align*}
\left[ P \oplus M_1 \oplus M_2 \right]_{0,2} & \supseteq \left[ P \right]_{0,2} \]

\[ \mathcal{L}_1[i] \quad \mathcal{L}_2[i] \quad R \quad M \quad M' \quad \mathcal{L}_1[i] \quad \mathcal{L}_2[i] \]

**CompCertX**

\[ \text{CompCertX( } M \text{) = } M' \]

\[ \begin{align*}
\mathcal{L}_1[i] & \subseteq \mathcal{L}_2[i] \\
C & \subseteq \text{Asm} \quad L_1[i] \quad L_2[i]
\end{align*} \]

QED
Assembly Layers

- L2[i]
- R
- M'
- L1[i]
- Asm

Horizontal Composition
Vertical Composition
Parallel Composition

Software Scheduler

```c
void yield () {
    uint t = tid();
    ...
    enq (t, rdq());
    uint s = deq (rdq());
    ...
    context_switch (t, s)
}
```

Shared Queue Lib
Spinlock

Software Scheduler

<table>
<thead>
<tr>
<th>Thread1</th>
<th>f1</th>
<th>yield</th>
<th>f2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread2</td>
<td>g1</td>
<td>g2</td>
<td>yield</td>
</tr>
</tbody>
</table>

CPU'1
private mem

T1 T1 T2 T2 T2 T1
CompCertX + Algebraic Memory Model = Thread-safe Verified Compiler

Verification of a Concurrent OS Kernel

<table>
<thead>
<tr>
<th>Layer</th>
<th>Redaction proof</th>
<th>Code verification</th>
<th>Source code</th>
<th>Proof linking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trap handler</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Virtual machine manager</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proc management</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU0</th>
<th>CPU1</th>
<th>CPU2</th>
<th>CPU3</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>T1</td>
<td>T2</td>
<td></td>
</tr>
</tbody>
</table>

Intermediate layer interface for multi-threaded linking

<table>
<thead>
<tr>
<th>Layer</th>
<th>Redaction proof</th>
<th>Code verification</th>
<th>Source code</th>
<th>Proof linking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread linking</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inter-thread</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Layer</th>
<th>Redaction proof</th>
<th>Code verification</th>
<th>Source code</th>
<th>Proof linking</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Contribution Summary

Certified Concurrent Abstraction Layers

CertiKOS

Strategy Refinement

$\psi \leq_R \psi'$

Thread-safe CompCert

Case Study

Build a Certified System

Compiler

User Application

Inter-Process Communication

Scheduling Module

Thread Queue Module

Spin-lock Module

Keyboard Driver

CPU 0

Keyboard

Security

CPU 1

Device Driver [PLDI16'a]

External events

read/write

State

Log

Raw Device Obj

Driver Layers

Device

Logical CPU

Interrupt

iret

CPU i
Summary: The CertiKOS / DeepSpec Project

**Killer-app:** high-assurance “heterogeneous” systems of systems!

**Conjecture:** today’s PLs fail because they ignored OS, and today’s OSes fail because they get little help from PLs

**New Insights:**
- deepspec & certified abstraction layers;
- a unifying framework for composing heterogeneous components (via game semantics + linear logic connectives)

**Opportunities:**
- New certified system software stacks (CertiKOS ++)
- New certifying programming languages (DeepSEA vs. C & Asm)
- New certified programming tools
- New certified modeling & arch. description lang. (DeepSEA)
- We verify all interesting properties (correctness, safety, security, availability, …)