CertiKOS: A Breakthrough toward Hacker-Resistant Operating Systems

Zhong Shao
Yale University
January 25, 2018

Acknowledgement: Ronghui Gu, Newman Wu, Hao Chen, Jieung Kim, Jeremie Koenig, Vilhelm Sjoberg, Mengqi Liu, Lionel Rieg, Quentin Carbonneaux, Unsung Lee, Jiyong Shin, David Costanzo, Tahina Ramananandro, Hernan Vanzetto, Shu-Chun Weng, Zefeng Zeng, Zhencao Zhang, Liang Gu, Jan Hoffmann, Joshua Lockerman, and Bryan Ford. This research is supported in part by DARPA CRASH and HACMS programs and NSF SaTC and Expeditions in Computing programs.
Motivation

- Mobile
- Desktop
- Transportation
- Health
- Aviation
- Financial
- Cloud
- Applications
- OS
- Hardware
- Environment
Motivation

Crash

Mobile

Accident

Cloud

Life

Financial

Loss

Environment

Applications

OS

Hardware
Motivation

System Software Runs Everywhere

Untrusted
- Software errors
- $312B cost

Test
- No!
Motivation

“Program testing can be used to show the presence of bugs, but never to show their absence.”

— Edsger Dijkstra
Motivation

“Complete **formal verification** is the only known way to guarantee that a system is free of programming errors.

— seL4 [SOSP’09]

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

— NSF SFM Report[2016]
Motivation

Complete formal verification is the only known way to guarantee that a system is free of programming errors.

“Formal verification is the only way to ensure security and privacy in computer systems.”
—NSF SFM Report [2016]

Formal Verification

- mathematically prove
- program **meets** specification
- under **all** inputs
- under **all** execution
- rule out entire **classes** of attacks
Motivation

System Software Runs Everywhere

Untrusted
- Software errors
- $312B cost

Test
- No!

Formal Verification
- Challenges?
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

Asm
Challenges: Compositionality

A Complex System
Challenges: Compositionality

A Complex System

[Diagram of a complex system with nodes labeled as 'Verify']
Challenges: Compositionality

A Complex System

Verify

Verify

Verify

Verify

Verify

Verify

C

Compiler

Asm
Challenges: Compositionality

A Complex System

Complete Verification

C

Compiler

Asm
Challenges: Concurrency

- Fine-grained lock
- I/O concurrency
- Multi-thread
- Multiprocessor
Challenges: Concurrency
Challenges: Concurrency

CPU i

CPU j

Complete Verification
Challenges: New Domain

System

Verification

Huge gap
Challenges: New Domain

Huge gap
Contribution

Certified Abstraction Layers

CertiKOS

aim to solve all these challenges
Contribution

Certified Abstraction Layers

untangle

fine-grained lock

CPU i

CPU j
Contribution

Certified Abstraction Layers

- verify existing systems
- build the next generation system software designed to be reliable and secure
Contribution

Certified Abstraction Layers

verify existing systems

build certified system software
Contribution

Certified Abstraction Layers

verify existing systems

build certified system software

System

Verification

Huge gap
Contribution

Certified Abstraction Layers

- verify existing systems
- build certified system software

System Verification
Contribution

Certified Abstraction Layers

- verify existing systems
- build certified system software

Certified System Software
Contribution

Certified Abstraction Layers

\[
\begin{array}{c}
L_1 \\
R_1 \\
M_1 \\
L \\
\end{array}
\quad \oplus 
\quad
\begin{array}{c}
L_2 \\
R_1 \\
M_2 \\
L \\
\end{array}
\]

\[
\begin{array}{c}
L \\
R_0 \\
M_0 \\
L_0 \\
\end{array}
\]
Contribution

Certified Abstraction Layers

\[ L_1 \oplus L_2 \]
\[ R_1 \]
\[ M_1 \oplus M_2 \]
\[ L \]
\[ \oplus \]

\[ L \]
\[ R_0 \]
\[ M_0 \]
\[ L_0 \]
Certified Abstraction Layers

Contribution

CompCertX

Asm
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] [CCAL 2017] the first formally certified concurrent OS kernel with fine-grained locks 6.5k C&Asm, 2 py
Certified System Software

- functional correctness
- liveness
- no stack/integer/buffer overflow
- no race condition
Contribution

**mC2**

Diagram of mC2 with various components and connections, including Sync & Mutual Exclusion (FIFOBBQ, CV, ...), Per Core (VM Monitor, Timer, TCB, k_stack, k_context, Cur TID, PCPU), Per Thread (TCB, k_stack, k_context), and CPU (Core 0, LAPIC 0, Core 1, LAPIC 1, ..., Core 8, LAPIC 8). The diagram also includes Process, IPC, Lib Mem, ELF Ldr, VMM, Page Map, PMM, Alloc Tbl, Spin Locks, Ticket, MCS, Container, and various other components related to hardware, data, drivers, kernel modules, and use cases.
Contribution

mC2

6.1k LOC
C layers

CompCertX

400 LOC
Asm layers

machine-checkable proof
Some of the significant results that were accomplished using Coq are proofs for the four color theorem, the development of CompCert (a fully verified compiler for C), the development at Harvard of a verified version of Google’s software fault isolation, and most recent, the fully specified and verified hypervisor OS kernel CertiKOS.
Deployment

CertiKOS on **Landshark**, DARPA HACMS
Deployment

CertiKOS on Quadcopter
Case Study

Build a Certified System

- Compiler
- User Application
- Inter-Process Communication
- Scheduling Module
- Thread Queue Module
- Spin-lock Module
- Keyboard Driver
- Send
- CPU 0
- Keyboard
- CPU 1
Certified Sequential Layer [POPL’15]

certified objects

specification of modules to trust
Certified Sequential Layer [POPL’15]

abs-state

certified objects

specification of modules to trust
Certified Sequential Layer [POPL’15]

abs-state

primitives

certified objects

specification of modules to trust
Certified Sequential Layer

module $M$

memory

$L_1$
Certified Sequential Layer

$M$

$L_1$

$L_2$
Certified Sequential Layer

specification

implementation

$L_1$

$L_2$

$M$
Certified Sequential Layer

**Certified Sequential Layer**

**specification**

**implementation**

$L_2$

$M$

$L_1$
Example: Thread Queue

typedef struct tcb {
    state s;
    tcb *prev, *next;
} tcb;
tcb tcbp[1024];
tcb *tcbp[0], *tcbp[1], *tcbp[2];

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

tdq* td_queue;
Example: Thread Queue

```c
typedef struct tcb {
    state s;
    tcb *prev, *next;
} tcb;

tcb tcbp[1024];

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

tdq* td_queue;
```

Implementation
Example: Thread Queue

```c
typedef struct tcb {
    state s;
    tcb *prev, *next;
} tcb;

tcb tcbp[1024];

tcb tdq* td_queue;

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

implementation
Example: Thread Queue

typedef struct tcb {
    state s;
    tcb *prev, *next;
} tcb;

tcb tcbp[1024];

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

dq* td_queue;
Example: Thread 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;
}

implementation
Example: Thread 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

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

\[
L_2
\]

Definition \texttt{tcbp} := ZMap.t state.
Definition \texttt{td_queue} := List Z.
Example: Thread Queue

specification

\[ L_2 \]

\[ \text{tcbp}(0) \quad \text{tcbp}(1) \quad \text{tcbp}(2) \]

\[ s_0 \quad s_1 \quad s_2 \]

Definition `tcbp` := ZMap.t state.
Definition `td_queue` := List Z.
Example: Thread Queue

specification

\[ L_2 \]

\begin{align*}
\text{tcbp}(0) & \quad \text{tcbp}(1) & \quad \text{tcbp}(2) \\
\text{s0} & \quad \text{s1} & \quad \text{s2} \\
\end{align*}

\begin{align*}
\text{td\_queue} & \\
1 & :: & 0 & :: & 2 & :: & \text{nil} \\
\end{align*}

Definition \textbf{tcbp} := ZMap.t state.
Definition \textbf{td\_queue} := List Z.
Example: Thread Queue

specification

implementation

$L_2$

$M$

tcbp(0) tcbp(1) tcbp(2)

td_queue

head

tail

Example: Thread Queue

specification

\[
L_2
\]

\[
\begin{array}{cccc}
\text{tcbp}(0) & \text{tcbp}(1) & \text{tcbp}(2) \\
s0 & s1 & s2 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 0 & 2 & \text{nil}
\end{array}
\]

Function **dequeue** (q) :=

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

Coq
Example: Thread Queue

specification

\[ L_2 \]

tcbp(0) tcbp(1) tcbp(2)

\[ \text{td\_queue} \]

\[ \text{s0} \quad \text{s1} \quad \text{s2} \quad \text{1} \]

\[ \text{0} :: \text{2} :: \text{nil} \]

Function \texttt{enqueue} (q) :=

\[
\text{match q with}
\begin{align*}
\text{head} :: q' & \Rightarrow (q', \text{Some head}) \\
\text{nil} & \Rightarrow (\text{nil}, \text{None})
\end{align*}
\text{end.}
\]

\text{Coq}
Simulation Proof

specification

implementation

Program Context

Deep Specification

$L_2$

$M$

$R$

$L_1$
Deep spec $L_2$ captures all we need to know about $M$ over $L_1$.

Any property about $M$ can be proved using $L_2$ alone.

No need to look at $M$ again.
mCertiKOS

kernel code

seq machine
memory management

seq machine
certified sequential kernel
mCertiKOS

- Trap
- VM
- PM
- TM
- MM

Diagram:
- trap
- proc
- thread
- mem
- seq machine

Set operations:
- \( \bigoplus \)
mCertiKOS

certified hypervisor

- trap
- vm
- proc
- thread
- mem
- seq machine

Options:
- Trap
- VM
- PM
- TM
- MM
mCertiKOS 3k LOC [POPL’15] 1 person year

Can boot Linux as a guest

TSysCall Layer

(pe, ikern, ihost, ipt, AT, PT, ptp, pbit, kctxp, Htcbp, Htcp, cid, chanp, uctxp, npt, hctx, vmst)

<table>
<thead>
<tr>
<th>thread_wakeup/kill/sleep/yield</th>
<th>pt_read</th>
<th>get/set_utctx</th>
<th>palloc/free</th>
<th>cid_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>sys_chan_send/recv/wait/check</td>
<td>sys_yield</td>
<td>sys_get_exit_reason</td>
<td>sys_get_eip</td>
<td></td>
</tr>
<tr>
<td>sys_check_shadow/pending_event</td>
<td>sys_proc_create</td>
<td>sys_set_seg</td>
<td>sys_inject</td>
<td></td>
</tr>
<tr>
<td>sys_get_exit_io_width/port/rep/str/write/eip</td>
<td>sys_set_intercept_int</td>
<td>sys_npt_instr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmcbinit</td>
<td>pagefault_handler</td>
<td>sys_reg_get/set</td>
<td>sys_sync</td>
<td>sys_run</td>
</tr>
</tbody>
</table>

TSysCall Layer

(mm/proc/virt.abs)

vmcbinit | sys_run/exit | PageFault_Handler | sys_yield | sys_check/exit/sync/inject/set-chan (17) | mm/proc_prim

TTrap Layer

(mm/proc/virt.abs)

vmcbinit | vm_run/exit | get_arg/set_ret | sys_check/exit/sync/inject/set-chan (17) | mm/proc_prim
certified sequential kernel

- trap
- virt
- proc
- thread
- mem

seq machine

multicore machine  CPU0  CPU1  CPU2  CPU3
Concurrent Framework [OSDI’16]
Concurrent Framework [OSDI’16]

- trap
- virt
- proc
- thread-local machine
- thread
- mem
- spin-lock
- seq machine
- CPU-local machine
- multicore machine
Certified Concurrent Layers

local certified objects

\[ L_1 \]
Certified Concurrent Layers

atomic objects

logical log

a sequence of events
Certified Concurrent Layers

atomic objects

logical log

a sequence of events
Certified Concurrent Layers

atomic objects

logical log
a sequence of events

$L_1$
Certified Concurrent Layers

to share

$\mathcal{L}_1$
Certified Concurrent Layers

$L_1$  

fine-grained locking

$L_2$
Concurrent Framework

CPU-local machine  CPU0  CPU1  CPU2  CPU3

machine lifting

multicore machine  CPU0  CPU1  CPU2  CPU3
step 0: raw x86 multicore model

multicore machine

CPU0  CPU1  CPU2  CPU3
step 0: raw x86 multicore model

non-determinism

CPU0 → atom → share → CPU1

CPU1 → private → atom

logical log: 0.a, 1.a

multicore machine: CPU0, CPU1, CPU2, CPU3
step 0: raw x86 multicore model

non-determinism

multicore machine

CPU0  CPU1  CPU2  CPU3
step 0: raw x86 multicore model

non-determinism
step 1: logical **hardware** scheduler

**multicore machine**

- CPU0
- CPU1
- CPU2
- CPU3
step 1: logical **hardware** scheduler

$\mathcal{E}_{hs}$

logical log 

| 0 | 0.a | 1 | 1 | 1.a | 0 |

multicore machine | CPU0 | CPU1 | CPU2 | CPU3 |
step 1: logical **hardware** scheduler

\[ \forall E_{hs} \]
step 2: push/pull memory model

\[ \forall \mathcal{E}_{hs}, \text{machine with hardware scheduler} \]

multicore machine

CPU0 | CPU1 | CPU2 | CPU3
step 2: push/pull memory model

∀ \mathcal{E}_{hs} machine with hardware scheduler

multicore machine

CPU0  CPU1  CPU2  CPU3
step 2: push/pull memory model

CPU0 ➔ pull ➔ share

logical copy

shared mem

∀ E_{hs} machine with hardware scheduler

multicore machine

CPU0  CPU1  CPU2  CPU3
step 2: push/pull memory model

- CPU0 pulls a shared logical copy
- There is a race condition

∀ ℋₚₛ, machine with hardware scheduler

multicore machine: CPU0, CPU1, CPU2, CPU3
step 2: push/pull memory model

CPU0 ➔ pull ➔ share ➔ push

logical copy

shared mem

∀ 𝔼_{hs} machine with hardware scheduler

multicore machine  CPU0  CPU1  CPU2  CPU3
step 2: push/pull memory model

∀ Ε_{hs} machine with hardware scheduler

multicore machine | CPU0 | CPU1 | CPU2 | CPU3
step 3: environment context model

0.a

atom

private

CPU0

0

CPU1

1

1

0

E_{hs}

CPU3

CPU2

CPU1

CPU0

C0

C1

C2

C3

machine with push/pull model

machine with hardware scheduler

multicore machine

CPU0  CPU1  CPU2  CPU3
step 3: environment context model

The image depicts a diagram illustrating the flow of data between different processors in a multicore machine with push/pull model and machine with hardware scheduler. The diagram shows the sequence of steps involving processors CPU0 and CPU1, with data flowing from one to the other, indicating the context model for the environment.

- **Environment Context Model:**
  - CPU0 sends data to CPU1.
  - CPU1 sends data back to CPU0.

- **Multicore Machine:**
  - Contains processors CPU0, CPU1, CPU2, and CPU3.

- **Models:**
  - Machine with push/pull model.
  - Machine with hardware scheduler.

The diagram uses symbols and numbers to represent the flow and context, emphasizing the interaction and context-awareness in the system.
step 3: environment context model

\[ \forall \mathcal{E}_{hs} \]

machine with push/pull model

machine with hardware scheduler

multicore machine

CPU0  CPU1  CPU2  CPU3
step 3: environment context model

machine with push/pull model

machine with hardware scheduler

multicore machine

CPU0  CPU1  CPU2  CPU3
step 4: remove unnecessary interleaving

\[
\forall \mathcal{E}_{hs} \quad \text{machine with push/pull model}
\]

\[
\text{machine with hardware scheduler}
\]

\[
\text{multicore machine} \quad \text{CPU0} \quad \text{CPU1} \quad \text{CPU2} \quad \text{CPU3}
\]
step 4: remove *unnecessary* interleaving

shuffle

atom → pull → share → private → push

∀ E_h,s

CPU i machine | CPU j machine
machine with push/pull model
machine with hardware scheduler

multicore machine

CPU0 | CPU1 | CPU2 | CPU3
step 4: remove **unnecessary** interleaving

merge

![Diagram showing merge process with labels: atom, pull, share, private, push.]

∀ \mathcal{E}_{hs} machine with hardware scheduler

multicore machine

CPU i machine

CPU j machine

machine with push/pull model

machine with hardware scheduler

CPU0  CPU1  CPU2  CPU3
Machine Lifting

logical \log \begin{array}{cccccc}
0 & 0.a & 1 & 1 & 1.a & 0
\end{array}

- seq machine
  - CPU-local machine
  - CPU i machine
  - machine with push/pull model
  - machine with hardware scheduler
- seq machine
  - CPU-local machine
  - CPU j machine

\forall \mathcal{E}_{hs}
Case Study

Build a Certified System

Compiler

User Application

Inter-Process Communication

Scheduling Module

Thread Queue Module

Spin-lock Module

Keyboard Driver

Send

CPU 0

Keyboard

CPU 1

Security
Case Study

Build a Certified System

Compiler

User Application

Inter-Process Communication

Scheduling Module

Thread Queue Module

Spin-lock Module

CPU 0

Keyboard

Security

Send

CPU 1
Acquire Lock Specification

safely pull

logical copy
Acquire Lock Specification

safely pull

logical copy

pull will eventually return
Acquire Lock Specification

- Logical copy
- Mutual exclusion
- Liveness
Example: Ticket Lock

mutual exclusion + liveness

```c
void acq_lock (uint i) {
    uint64 t = FAI_ticket (i);

    while (get_now (i) != t) {
    }

    pull (i);
}
```
Example: Ticket Lock

mutual exclusion + liveness

```c
void acq_lock (uint i)
{
    uint64 t = FAI_ticket (i);
    while (get_now (i) != t)
    {
    }
    pull (i);
}
```
Example: Ticket Lock

mutual exclusion + liveness

```c
void acq_lock (uint i)
{
    uint64 t = FAI_ticket (i);
    while (get_now (i) != t)
    {
        pull (i);
    }
}
```
Example: Ticket Lock

mutual exclusion + liveness

```c
void acq_lock (uint i)
{
    uint64 t = FAI_ticket (i);
    while (get_now (i) != t)
    { }
    pull (i);
}
```
Example: Ticket Lock

mutual exclusion + liveness

```c
void acq_lock (uint i)
{
    uint64 t = FAI_ticket (i);
    while (get_now (i) != t)
    {
    }
}
```

Example: Ticket Lock

mutual exclusion + liveness

```c
void acq_lock (uint i) {
    uint64 t = FAI_ticket (i);
    while (get_now (i) != t) {} 
    pull (i);
}
```
Example: Ticket Lock

**mutual exclusion + liveness**

```c
void acq_lock (uint i)
{
    uint64 t = FAI_ticket (i);
    while (get_now (i) != t)
    {
    }
    pull (i);
}
```

#CPUs < 2^{64}
Example: Ticket Lock

mutual exclusion + liveness

```c
void acq_lock (uint i) {
    uint64 t = FAI_ticket (i);
    while (get_now (i) != t) {
    }
    pull (i);
}
```

#CPUs is bounded
a fair hardware scheduler
lock holders will release lock
Example: Ticket Lock

```
acq_lock
acq lock

acq_lock
FAI ticket get now get now pull
```
Example: Ticket Lock

```c
void acq_lock (uint i) {
    uint64 t = FAI_ticket (i);
    while (get_now (i) != t) {}  // bug in the original implementation
    pull (i);  // mutual exclusion will be violated when there is an integer overflow for t
}
```
Case Study

Build a Certified System

Compiler

User Application

Inter-Process Communication

Scheduling Module

Thread Queue Module

Spin-lock Module

Keyboard Driver

Send

CPU 0

Keyboard

Security

CPU 1
Case Study

Build a Certified System

Compiler

User Application

Inter-Process Communication

Scheduling Module

Thread Queue Module

Spin-lock Module

Keyboard

CPU 0

Keyboard

CPU 1

Send

Security
Example: Shared Thread Queue

dequeue

dequeue

local memory
Example: Shared Thread Queue

dequeue

local memory
Example: Shared Thread Queue

logical copy

shared memory

acq lock

dequeue
Example: Shared Thread Queue

acq lock

dequeue

logical copy

shared memory
Example: Shared Thread Queue

logical copy

shared memory

acq lock    dequeue    rel lock
Example: Shared Thread Queue

logical copy

shared memory

acq lock dequeue rel lock
Example: Shared Thread Queue

- dequeue
- deq

shared memory
Example: Shared Thread Queue

shared memory

dequeue
Build a Certified System

Case Study

Compiler

User Application

Inter-Process Communication

Scheduling Module

Thread Queue Module

Spin-lock Module

Keyboard Driver

CPU 0

Keyboard

Security

CPU 1

Send
Case Study

Build a Certified System

Compiler

User Application

Inter-Process Communication

Scheduling Module

Thread Queue Module

Spin-lock Module

Keyboard Driver

Send

CPU 0

Keyboard

Security

CPU 1
Thread-Local Machine

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

Found hard bugs in the popular OS textbook
[Operating Systems Principles and Practice 2011]

Software Scheduler

close-up

sleep  yield  wakeup
Case Study

Build a Certified System

Compiler

User Application

Inter-Process Communication

Scheduling Module

Thread Queue Module

Spin-lock Module

Keyboard Driver

Send

CPU 0

Keyboard

CPU 1

Security
Build a Certified System

- Compiler
- User Application
- Inter-Process Communication
- Scheduling Module
- Thread Queue Module
- Spin-lock Module

Send

Keyboard Driver

CPU 0

Security

CPU 1

Case Study
Device Driver [PLDI16’a]
Device Driver [PLDI16’a]
Case Study

Build a Certified System

Compiler
User Application
Inter-Process Communication
Scheduling Module
Thread Queue Module
Spin-lock Module
Keyboard Driver
Send
CPU 0
Keyboard
CPU 1
Security
Build a Certified System

User Application
- Inter-Process Communication
- Scheduling Module
- Thread Queue Module
- Spin-lock Module

Compiler

Keyboard Driver

Send

CPU 0

Keyboard

CPU 1

Security

Case Study
End-to-End Security [PLDI16’b]

Observation function $O$
- specify and prove general security policies with declassification
- security-preservation simulation
- non-interference
- found security-bugs: spawn, palloc, …
Case Study

Build a Certified System

Compiler

User Application

Inter-Process Communication

Scheduling Module

Thread Queue Module

Spin-lock Module

Keyboard Driver

Send

CPU 0

Keyboard

Security

CPU 1
CertiKOS is the first fully certified OS kernel that is done economically (< 3 person years), proves more properties, runs on concurrent HW, and is truly extensible.

Still very high barriers of entry:

1. OS kernel development is very difficult
2. Formal specifications and proofs are hard to build
3. Need intimate programming language expertise to succeed
   - These are three completely different communities
   - Most people can only do one out of the above three.
   - The Yale team has been working on all three for >15 years.
**Summary: OS Landscape (Nov 2017)**

- **Desktop:** Linux, macOS, Windows, ChromeOS, freeBSD, …
- **Hypervisor/Cloud:** Linux KVM & Docker, VMWare, Xen, …
- **Mobile:** Android (Linux), iOS, …
- **Embedded:** Embedded Linux, VxWorks, QNX, LynxOS, …

- All of them are bloated, old, and contain many bugs
- Urgently need new OSes for emerging platforms & apps (IoTs, Drones, Self-Driving Cars, Cloud, NetworkOS, Blockchains, …)

**OS evolution has reached an inflection point:**
Need a certified OS that provides security, extensibility, performance, and can work across multiple platforms.