

Lecture 01: Formal Requirements Requirements for Virtualizable Virtualizable Architectures

Based on “Formal Requirements for Virtualizable Third Generation Architectures”
Architectures” Gerald J. Popek and Robert P. Goldberg — Communications of the
Communications of the ACM, volume 17, number 7, July 1974

Contents

01 The 1974 problem

02 A model of third generation machines

03 Instruction classes

04 The virtual machine monitor

05 Theorem 1

06 Recursion and hybrids

07 Limits of the model

08 The theorem after 1974

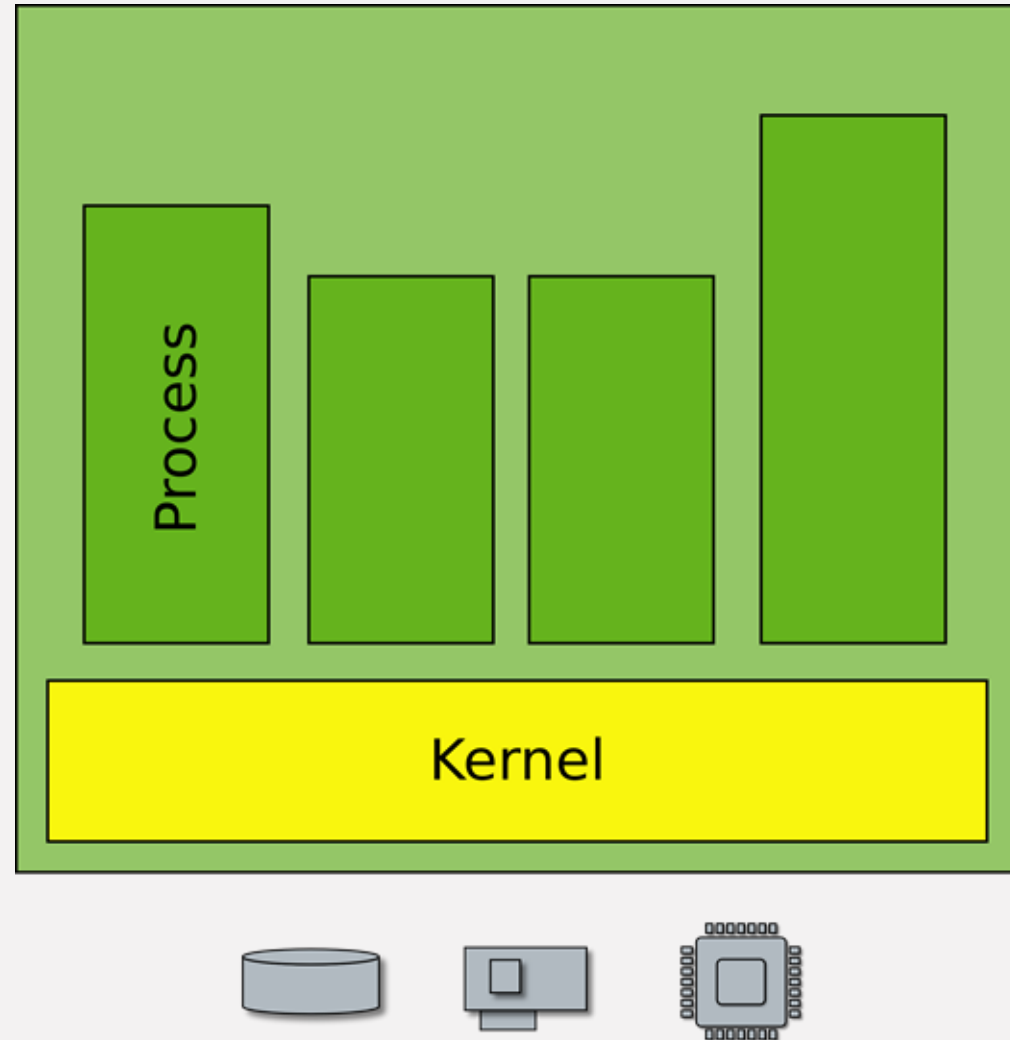
01

The 1974 problem

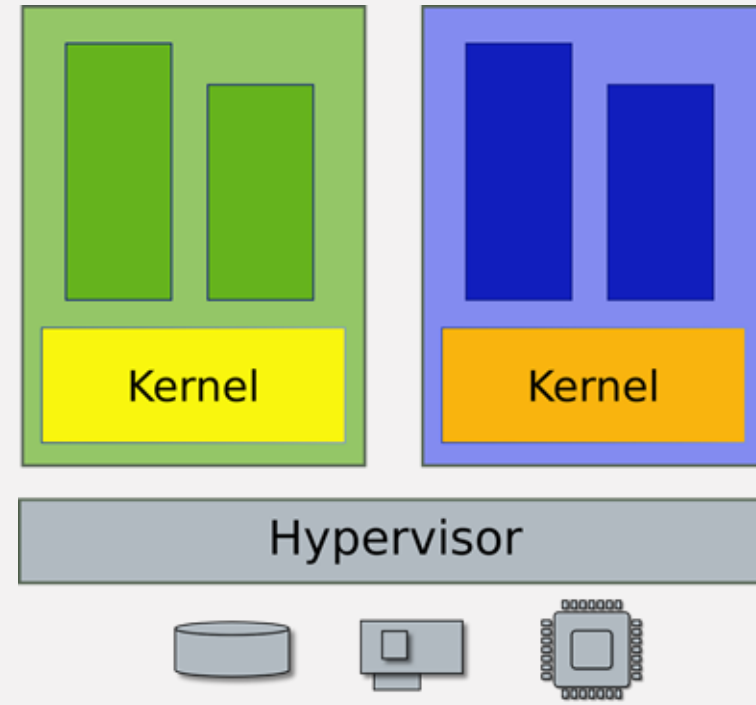
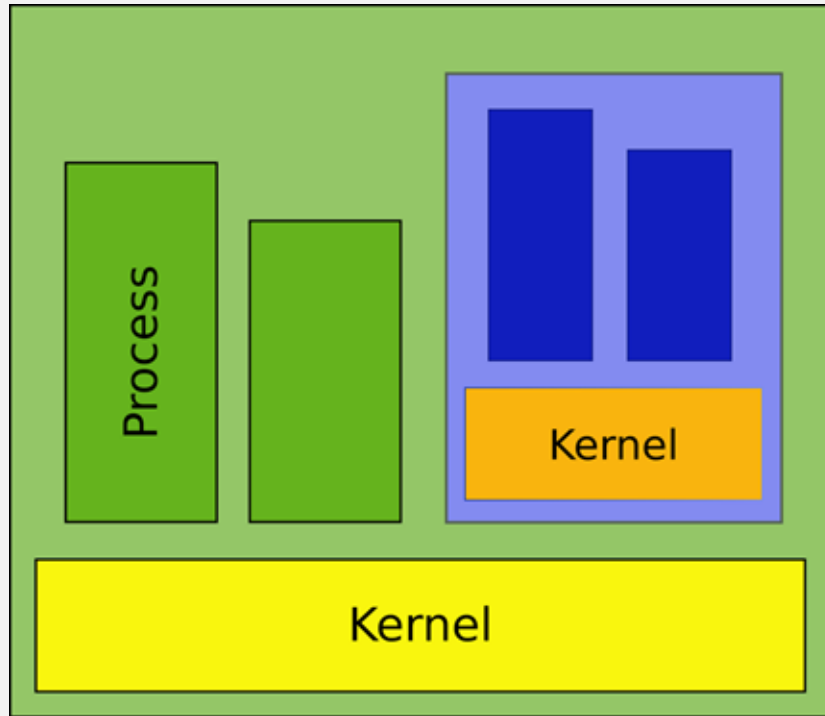
**“A virtual machine is taken to be an efficient,
isolated duplicate of the real machine.”**

— Popek and Goldberg, section 1

Traditional operating system



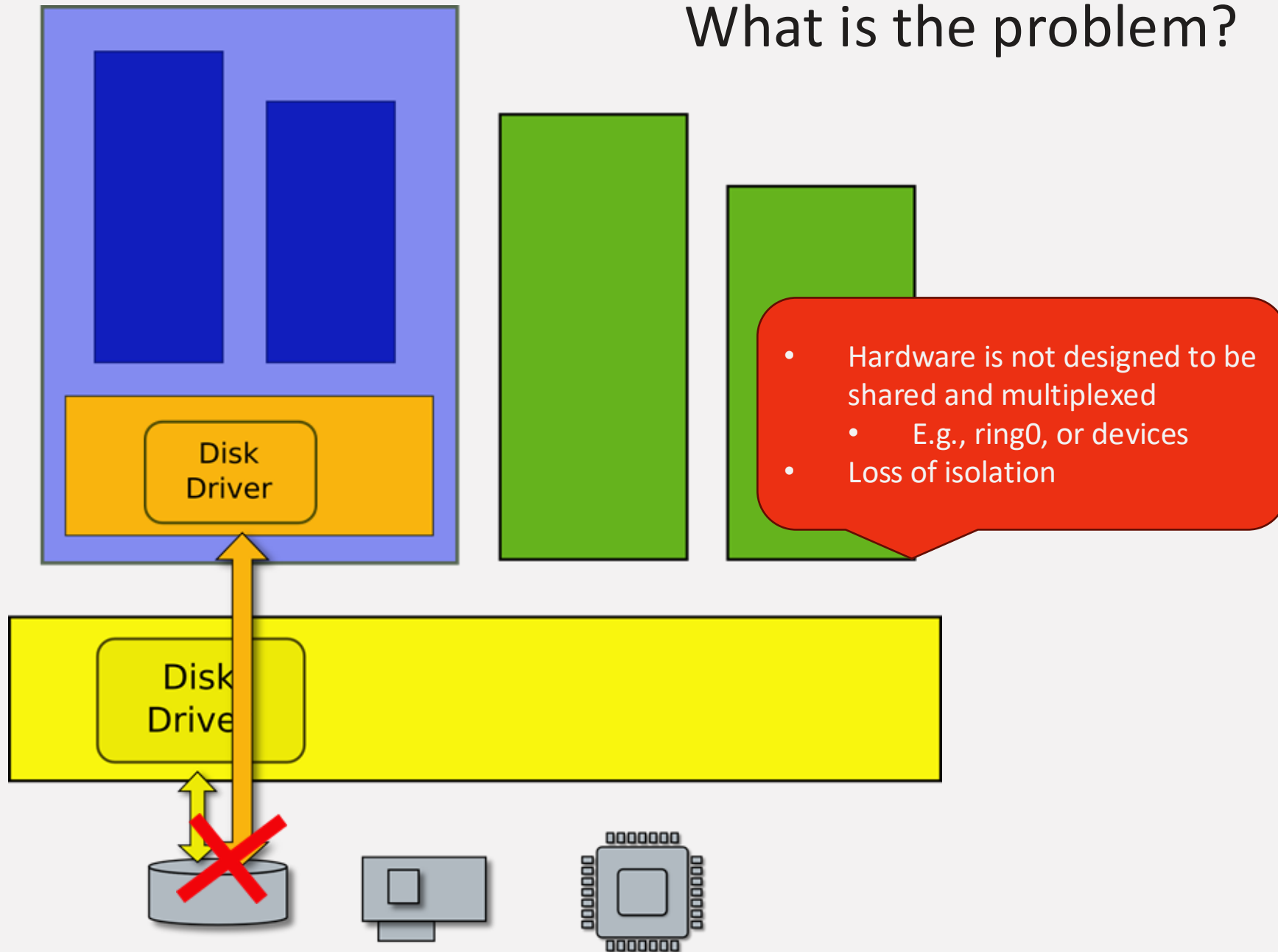
Virtual machines



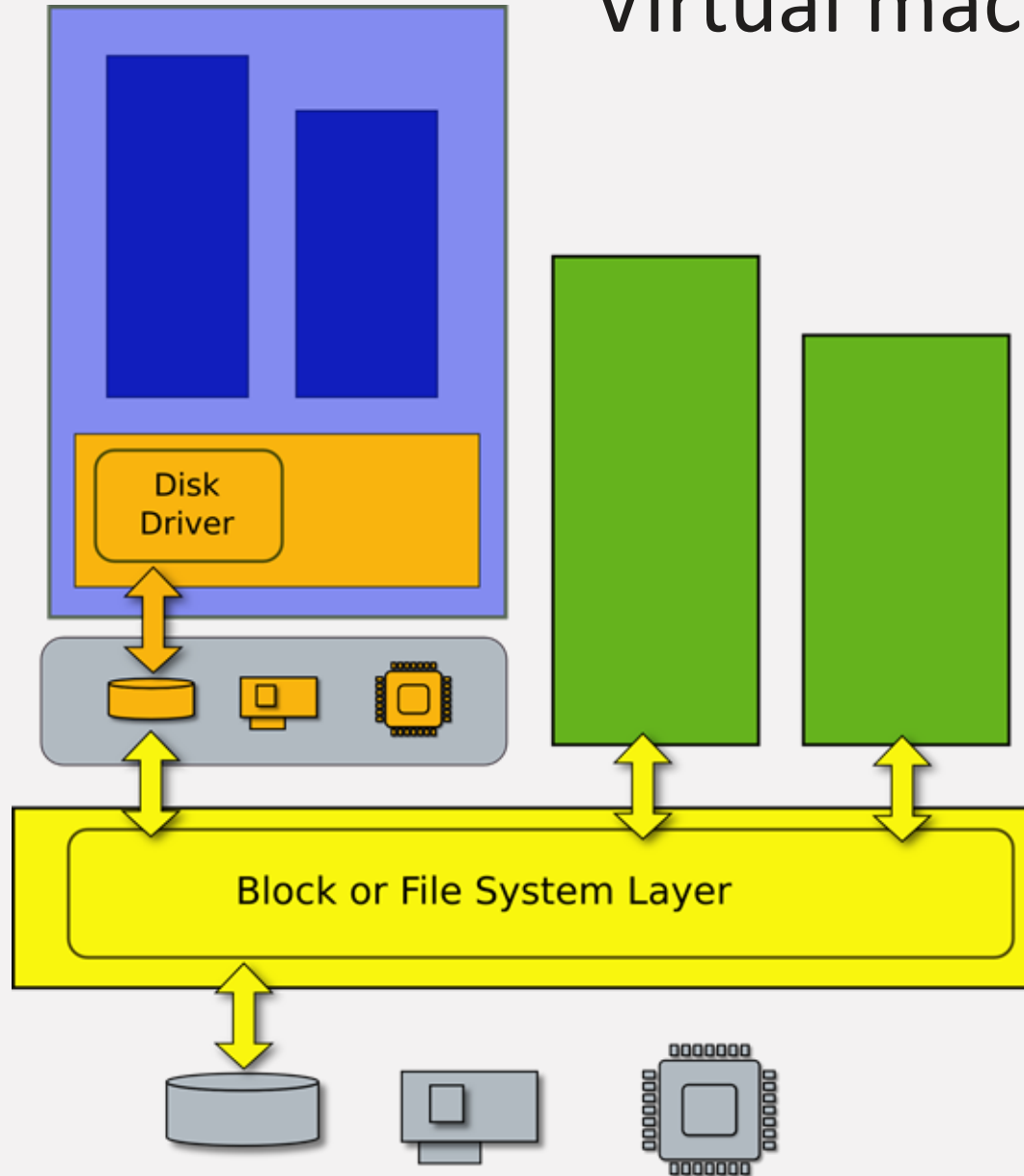
A bit of history

- Virtual machines were popular in 60s-70s
 - Share resources of mainframe computers [Goldberg 1974]
 - Run multiple single-user operating systems
- Interest is lost by 80s-90s
 - Development of multi-user OS
 - Rapid drop in hardware cost
 - Hardware support for virtualization is lost

What is the problem?

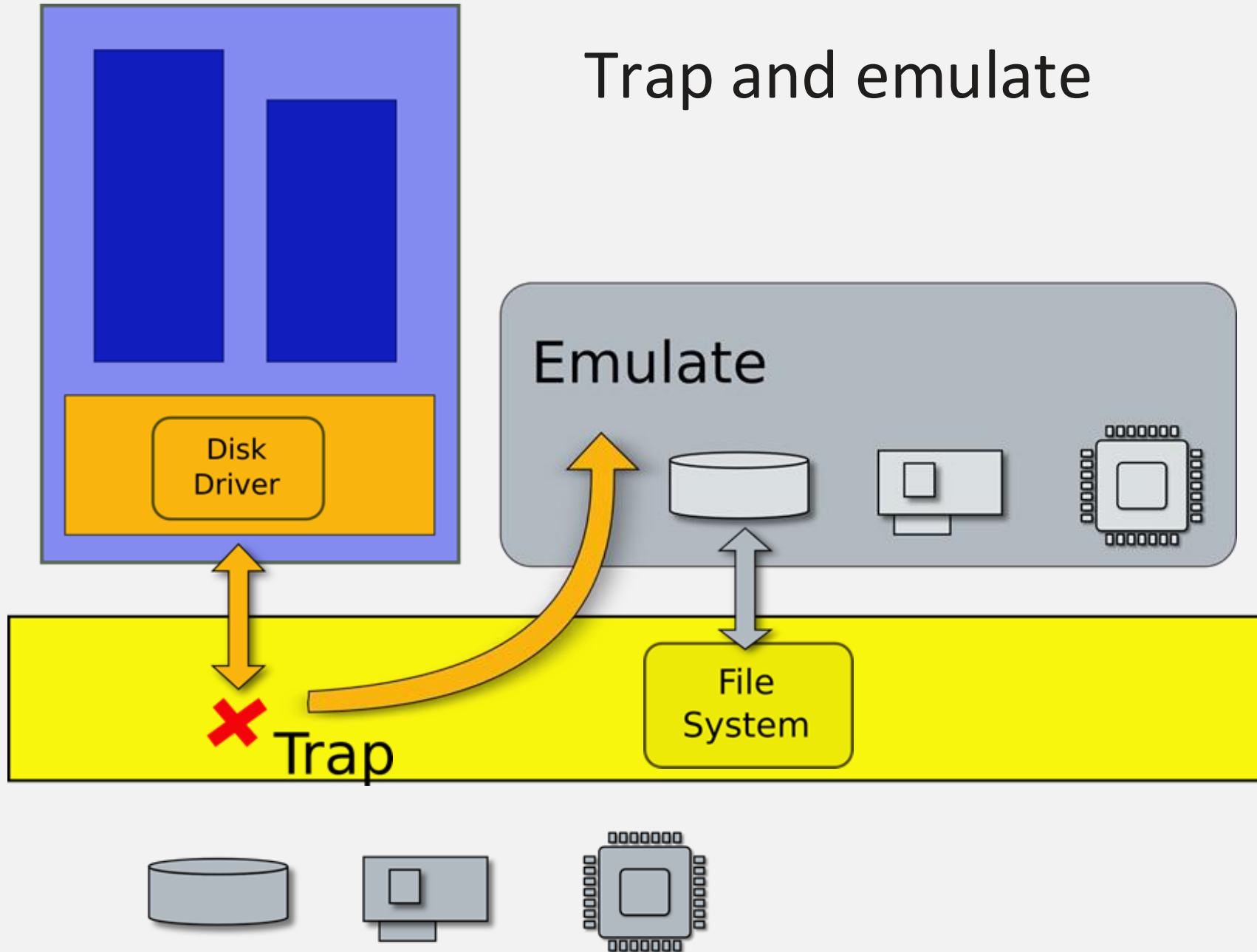


Virtual machine



- Efficient duplicate of a real machine
- Compatibility
- Performance
- Isolation

Trap and emulate



What needs to be emulated?

- CPU and memory
 - Register state
 - Memory state
- Memory management unit
 - Page tables, segments
- Platform
 - Interrupt controller, timer, buses
 - BIOS
- Peripheral devices
 - Disk, network interface, serial line

Three characteristics of a monitor

ONE

An identical environment

Programs see an environment essentially identical with the original machine, excepting excepting differences of resource availability availability and timing.

TWO

At worst a minor slowdown

A statistically dominant subset of the virtual processor's instructions executes directly on the real processor, with no software intervention.

THREE

Complete resource control

Memory and peripherals are the monitor's to allocate. Processor activity, notably, is not counted as a resource.

Complete control, in two clauses

CLAUSE ONE · CONFINEMENT

It is not possible for a program running in the created environment to access any resource not explicitly allocated to it.

CLAUSE TWO · REVOCABILITY

It is possible, under certain circumstances, for the monitor to regain control of resources already allocated.

What the definition excludes

A time-sharing system

Offers an extended machine, not an identical one, so it fails the identity clause.
The paper names this exclusion explicitly.

Emulators and simulators

Interpret everything, so nothing runs directly on the real processor. Fails the efficiency clause outright.

Virtual memory alone

One ingredient, not the whole environment.
The virtual machine has a virtual processor and possibly other virtual devices too.

02

A model of third generation machines

The machine

- A processor with two modes, supervisor and user. In user mode the full instruction repertoire is not available.
- Linear, uniformly addressable memory, addressed relative to a relocation-bounds register that is always active.
- The usual repertoire: arithmetic, testing, branching, moving data.
- The table look-up and copy property — a lookup on a table of arbitrary size, key and value, and the value moved anywhere in memory.

The state

$S = (E, M, P, R)$

SYMBOL	COMPONENT	CONTENTS
E	Executable storage	Word or byte addressed memory of size q; E[i] is the ith unit
M	Processor mode	Either s or u — supervisor or user
P	Program counter	An address relative to R, indexing the next instruction
R	Relocation-bounds register	A pair (l, b): base address l, absolute size b

- The triplet (M, P, R) is the program status word. An instruction is a function $i : C \rightarrow C$.

Address development

For an instruction producing address a :

if $a + I \geq q$ then memorytrap

else if $a \geq b$ then memorytrap

else use $E[a + I]$

- To address all of memory, relocation is set to 0 and bounds to $q - 1$. Every definition that follows is stated only for executions that do not memory trap.

Traps

DEFINITION · AN INSTRUCTION TRAPS

$i(E_1, M_1, P_1, R_1) = (E_2, M_2, P_2, R_2)$ where $E_2[j] = E_1[j]$ for $0 < j < q$, and $E_2[0] = (M_1, P_1, R_1)$, and $(M_2, P_2, R_2) = E_1[1]$.

- Storage is left unchanged except for location zero, which receives the old PSW. The new PSW is taken from location one. In most third generation software $M_2 = s$ and $R_2 = (0, q - 1)$.
- The definition may be relaxed: the trap need not block the instruction, provided the state is saved so as to be reversible to the point at which the instruction was about to execute.

The trap sequence

01



Issue

i attempted, mode = u
R active, P relative

02



Trap

storage unchanged,
except location zero

03



Exchange

$E[0] \leftarrow \text{old PSW}$
 $\text{new PSW} \leftarrow E[1]$

04



Dispatch

mode = s, R = (0, q-1)
P at the dispatcher

- Every instruction the monitor must not let a guest execute directly has to arrive at station two. Theorem 1 gives the condition under which that can be arranged.

Deliberate omissions

- No I/O instructions and no devices. The paper's cover is the PDP-11, where devices are memory cells.
- No interrupts, and so no asynchrony anywhere in the model.
- The processor is not treated as a resource — which is why HALT may execute directly.
- No isolated condition codes: instructions interact only through the PSW.

03

Instruction classes

Privileged instructions

DEFINITION

i is privileged if and only if for any pair of states $S_1 = (e, s, p, r)$ and $S_2 = (e, u, p, r)$ in which neither $i(S_1)$ nor $i(S_2)$ memory traps, $i(S_2)$ traps and $i(S_1)$ does not.

IN PLAIN TERMS

Same storage, same counter, same relocation — only the mode differs. From user mode the instruction traps; from supervisor mode it runs. Privilege is a property of the machine, independent of virtualization, readable from the principles of operation.

- The definition requires a trap. An instruction that merely does nothing in user mode is not privileged — the paper's own suggestion is “user mode NOP”. Remember the phrase.

Privileged instructions in three machines

PATTERN	MACHINE	INSTRUCTION	ALSO CONTROL SENSITIVE
if M = s then load_PSW else trap	IBM System/360	LPSW	Yes
if M = s then load_R else trap	Honeywell 6000	LBAR	Yes
if M = s then load_R else trap	DEC PDP-10	DATAO APR	Yes

- Both patterns touch the program status word or the relocation register — the two things the model treats as control. The era's machines protected the obvious cases.

Control sensitive instructions

DEFINITION

i is control sensitive if there exists a state $S_1 = (e_1, m_1, p_1, r_1)$ with $i(S_1) = S_2 = (e_2, m_2, p_2, r_2)$ such that $i(S_1)$ does not memory trap and either $r_1 \neq r_2$, or $m_1 \neq m_2$, or both.

IN PLAIN TERMS

The instruction attempts to change the memory resources available, or the processor mode, without going through the memory trap sequence. One such state is enough — the definition is existential.

- Example: JRST 1 on the DEC PDP-10, return to user mode. It changes M , so it is control sensitive; it is legal from supervisor mode, so it is not privileged.

Notation for relocation

$$r \oplus x = (l + x, b)$$

$E \mid r$ = the contents of memory from l to $l + b$

$$E \mid r = E' \mid r \oplus x \iff E[l + i] = E'[l + x + i] \text{ for } 0 \leq i < b$$

- The shift moves the base and leaves the bounds alone. Only the window $E \mid R$ is reachable from a state, so it is the only part of storage the state description needs. Strictly, the restricted form denotes an equivalence class of states, which instructions preserve.

Behavior sensitive instructions

DEFINITION

i is behavior sensitive if there exist an integer x and states $S_1 = (e \mid r, m_1, p, r)$ and $S_2 = (e \mid r \oplus x, m_2, p, r \oplus x)$, with $i(S_1) = (e_1 \mid r, m_1, p_1, r)$ and $i(S_2) = (e_2 \mid r \oplus x, m_2, p_2, r \oplus x)$, neither memory trapping, such that either $e_1 \mid r \neq e_2 \mid r \oplus x$, or $p_1 \neq p_2$, or both.

- The same program at the same relative position, shifted by x in physical memory. If the visible result differs, the instruction's effect depends on its location or its mode.
- If instead R or M came out different, that would be control sensitivity. Behavior sensitivity is about the result differing, not the control state.
- The state has no general-purpose registers: a fifth component G would add one disjunct, $g_1 \neq g_2$.

Location and mode sensitivity

SPECIES ONE

Location sensitive

Execution behavior depends on the instruction's location in real memory. The paper's example is load physical address, ERA on the IBM 360/67, which hands a program its own physical address.

SPECIES TWO

Mode sensitive

Behavior is affected by the machine's mode. The example is move from previous instruction space, MVPI on the DEC PDP-11/45, which forms its effective address from mode-dependent information.

- Location sensitivity has an escape hatch the paper admits later: a monitor resident in high core, with everything else running unrelocated, never moves anything — so location cannot be observed.

Location sensitivity, traced

THE SAME PROGRAM	PLACED AT L = 0	RELOCATED BY X = 4096
Relocation register R	(0, b)	(4096, b)
Apparent address it asks about	100	100
ERA writes back	100	4196
What the program concludes	It sits at zero	It has been moved

- Program image, relative counter and bounds are identical; only the base differs. So ERA is location sensitive, and a monitor must intercept it and substitute 100.

Mode sensitivity, traced

THE SAME GUEST CODE	ON THE REAL MACHINE	UNDER A MONITOR
Hardware mode M	s	u
Mode the guest believes it has	s	s — held by the monitor
What the instruction reports	s	u
What the guest concludes	It is privileged	It is not

- R is identical in both columns and neither execution changes R or M. Nothing about the control state moved — only the reported result differs, which is precisely behavior sensitivity.

Sensitive and innocuous

sensitive = control sensitive $\cup$ behavior sensitive

innocuous = everything else

- Innocuous instructions are defined by negation, and they are the great majority — the ones a monitor wants running directly on the hardware. The question the rest of the paper answers is what happens when the sensitive set is contained in the privileged set.

04

The virtual machine monitor

The control program

$$CP = (D, A, \{v_i\})$$

- The control program runs in supervisor mode; everything else runs in user mode. One location inside the control program therefore has to record the virtual machine's simulated mode — a small detail that Lemma 2 depends on.

Dispatcher, allocator, interpreters

D

Dispatcher

Its first instruction sits where the hardware traps — the P in location one. The top-level control module: it decides which module to call. Several trap vectors simply mean several entry points.

A

Allocator

Decides what system resources are provided. With one guest, it keeps guest and monitor apart. With several, it also prevents the same memory being given to two guests at once.

$\{V_i\}$

Interpreters

One routine per privileged instruction, each simulating the effect of the instruction that trapped. Indexed by privileged instructions — not by sensitive ones.

The three properties

ONE

Efficiency

All innocuous instructions are executed by the hardware directly, with no intervention at all on the part of the control program.

TWO

Resource control

It must be impossible for the program to affect the system resources available to it. The allocator is to be invoked upon any attempt.

THREE

Equivalence

Any program performs indistinguishably from the case where the control program did not exist — with two exceptions.

- A virtual machine monitor is defined as any control program satisfying all three. The environment such a program produces is a virtual machine.

The two exceptions to equivalence

EXCEPTION ONE

Timing

Because the control program occasionally intervenes, some instruction sequences take longer to execute, so assumptions about execution time may lead to incorrect results. The paper assumes, for the time being, that there are no such difficulties — and never returns to discharge the assumption.

EXCEPTION TWO

Resource availability

The allocator may not satisfy a request for space, since the control program itself takes room. The program may then be unable to function as it otherwise would. This exception the paper does resolve.

A smaller machine, logically the same

- The environment produced is a smaller version of the actual hardware: logically the same, with a lesser quantity of certain resources.
- Equivalence is then guaranteed against a real machine of that smaller size — not against the full machine.
- On a paged machine the resource consumed is more likely drum space to hold the monitor's pages.
- This is also what makes recursion cheap: a monitor can run inside a logically identical smaller machine.

05

Theorem 1

THEOREM 1

For any conventional third generation computer, a virtual machine monitor may be constructed if the set of **sensitive instructions for that computer is a subset of the set of **privileged** instructions.**

- Every other assumption — relocation, two modes, traps, a general enough instruction set — was already standard practice. The containment is the only new constraint: modest, easy to check against a manual, and usable as a design requirement.

The partition of states

C_v

With a monitor

States in which the monitor is present in memory and the P in the PSW stored in location one equals the monitor's first location. Defined by the trap vector, not by what is currently executing — a guest in user mode is still in this set.

- Each instruction is a unary operator on states, and so is each finite instruction sequence. Call the set of all finite sequences I .

C_r

Without a monitor

All remaining states: the possible states of the real machine with no monitor installed. The proof will map from here into the other set and show that the mapping commutes with execution.

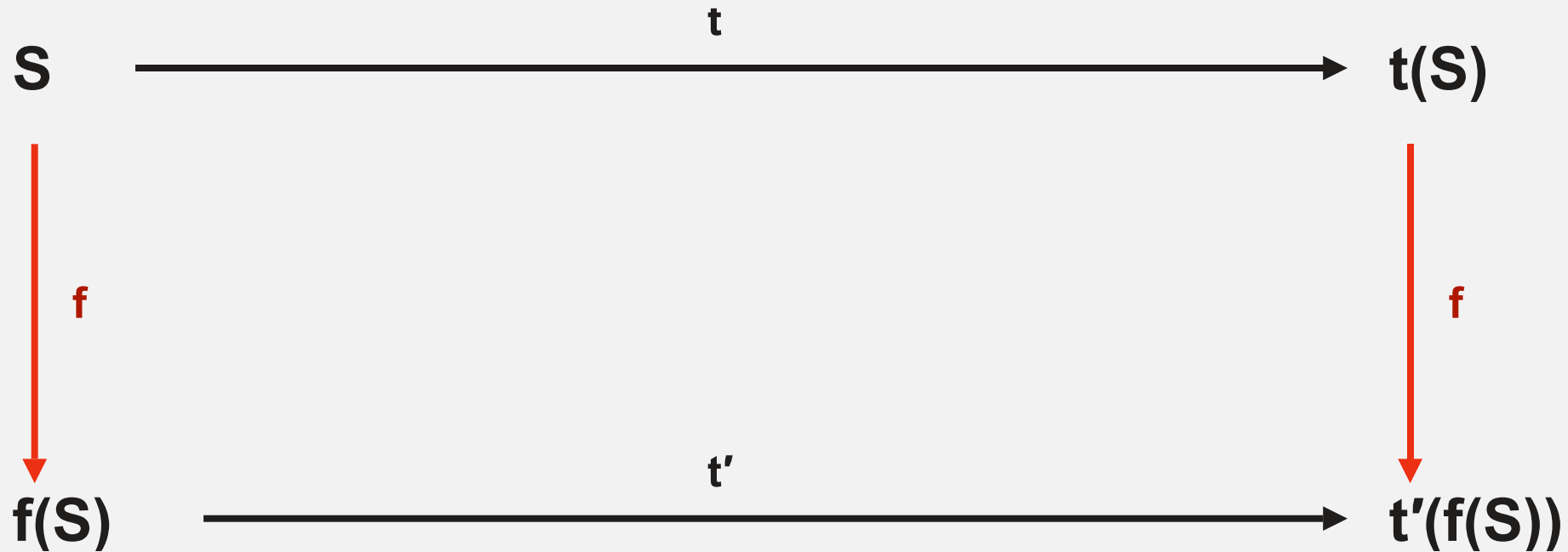
The VM map

THE MAP f

A one-to-one homomorphism from states of the bare machine into states of the machine with a monitor resident. For $S' = f(S)$: the visible memory window and the program counter agree, and the bounds part of the relocation register agrees.

- The mode need not agree. The guest's supervisor mode is simulated: the real machine runs in user mode while the monitor holds the simulated mode value in one of its own locations. This is exactly what Lemma 2 uses.
- The paper never writes f out explicitly, and does not need to. Only these properties are used in the lemmas.

The equivalence to be shown



- For any finite sequence t and any state S : $f(t(S)) = t'(f(S))$. The primed sequence is t with interpretation sequences spliced in where the sensitive instructions were — which is why it is not t .

Constructing the interpreters

- The effect of any instruction depends only on M , P , R , location one and the window E | R — not on all of memory.
- So tabulate it: for each privileged instruction, a table of pairs mapping each such restricted state to its successor.
- The tables are enormous. They can be made small only by choosing w , the maximum window size, small — that is, by limiting the size of the real machine.
- The table look-up and copy property then guarantees a third generation machine can manage them.
- The paper's own words: this is an existence argument only, such tables are a last resort, and in all real cases much simpler and more efficient routines exist and should be used.

Resource control and efficiency

PROPERTY TWO

Resource control

Any instruction that would affect resource allocation is control sensitive, hence by hypothesis privileged, hence it traps and passes control to the monitor. This chain is where the theorem's condition is spent.

- Only equivalence remains, and it takes three lemmas.

PROPERTY ONE

Efficiency

Efficiency was defined as direct execution of innocuous instructions, and the monitor was constructed to leave them alone. It holds by construction — which is also to say it was defined into existence rather than measured.

Lemma 1 — innocuous instructions

LEMMA 1

Innocuous instructions, as executed by the virtual machine system, obey the equivalence property.

- Let i be innocuous, S any real state, $S' = f(S)$. From the definition of f the visible window, the program counter and the bounds all agree. From the definition of innocuous, i 's effect cannot depend on m or on l . All other parameters agree, so $i(S) = i(S')$.
- This is what the behavior-sensitivity notation was for: it lets the whole dependence on mode and location be dismissed in one clause.

Lemma 2 — sensitive instructions

LEMMA 2

Sensitive instructions, as interpreted by the virtual machine system, obey the equivalence property.

- By assumption any sensitive instruction is privileged, so it traps. By construction the interpretation is correct given its parameters. The values in $E \mid R$ are not changed by the trap; P and R are saved in $E[0]$; the simulated mode is stored by the monitor. All necessary information is present.
- Three earlier decisions pay off at once: the trap preserves storage, the old PSW lands in location zero, and the monitor keeps the simulated mode. None looked important when introduced.

Lemma 3 — finite sequences

LEMMA 3

Given that all single instructions obey the equivalence property, any finite sequence of instructions also obeys it.

- By induction on sequence length. The basis is Lemmas 1 and 2. The primed operators may not be the same instructions or sequences as the unprimed ones, since the primed side carries the interpretation sequences — so the induction is over the correspondence, not over identity.
- This is where “indistinguishable” is finally earned. Equivalence was a claim about programs, and programs are finite instruction sequences ending in a halt. The full algebra is in the appendix.

Why the converse fails

- The condition is sufficient, not necessary. The paper says so and gives two escapes.
- Location sensitive instructions stop mattering if the monitor resides in high core and everything else runs unrelocated — nothing is moved, so location is unobservable.
- An instruction may not be privileged and still trap whenever an undesirable action would result: one that can change the relocation-bounds register but can only decrease the bounds from user mode.
- Both escapes work by knowing something extra about intended usage. Every practical workaround since has the same shape.

06

Recursion and hybrids

THEOREM 2

A conventional third generation computer is **recursively virtualizable if it is virtualizable, and a virtual machine monitor without any timing dependencies can be constructed for it.**

- The proof: a monitor produces an environment where a large class of programs runs identically, and a monitor belongs to that class. The excluded programs are the resource bound and the timing dependent. Timing is excluded by hypothesis (b) — the section 4 assumption, promoted to a stated condition. Memory boundedness only limits the depth of nesting, which the definition already allows for.

User sensitive and supervisor sensitive

USER SENSITIVE

Trouble from user mode

There exists a state $S = (E, u, P, R)$ for which i is control sensitive or behavior sensitive. These are the instructions that cause difficulty when executed from user mode — where guest application code runs.

SUPERVISOR SENSITIVE

Trouble from supervisor mode

There exists a state $S = (E, s, P, R)$ for which i is control sensitive or behavior sensitive. Guest supervisor code is a minority of a typical workload, so interpreting all of it is affordable.

- The two subsets are not necessarily disjoint. The paper is explicit about that.

THEOREM 3

A **hybrid virtual machine monitor may be constructed for any conventional third generation machine in which the set of **user sensitive** instructions are a subset of the set of privileged instructions.**

- In a hybrid monitor, all instructions in virtual supervisor mode are interpreted; otherwise it is the same object. Less efficient, and applicable to more machines. The paper's justification is blunt: there exist very few third generation architectures which are virtualizable.

07

Limits of the model

What the model sets aside

ONE

I/O

Justified by one machine. But DMA bypasses relocation, device state often cannot be read back, and channel programs are themselves programs. Every practical difficulty lives here.

TWO

Interrupts

No asynchrony at all, so the question of when a trap is delivered relative to guest-visible state never arises. Asynchronous I/O completion sits in the intersection of these first two.

THREE

Condition codes

Instructions interact only through the PSW. On real machines, arithmetic exception traps have to be excluded from control sensitivity by hand — a patch, not a principle.

- The paper says the requirements arising from I/O are “of a very similar nature.” That claim is asserted, never shown, and it is the largest unpaid debt in the paper.

Efficiency without a timing model

- Efficiency is stated qualitatively: innocuous instructions execute directly. There is no cost model and no bound on trap frequency.
- So a machine can satisfy Theorem 1 and still be useless — a monitor that is formally efficient and practically an interpreter.
- Timing was excluded from equivalence by assumption, never discharged, then promoted to a hypothesis in Theorem 2.
- Time is the blind spot in both directions: the model cannot say how fast a monitor is, nor handle a guest that can tell.

Sufficient, not necessary

- The theorem gives a test that can say yes. When it says no, almost nothing has been learned.
- The paper's own escapes prove the point, and thirty years of x86 work proves it at length.
- No companion necessary condition appears here, and no general and useful one has appeared since.
- Every escape depends on facts about intended usage, not about the architecture — which may be why.

The state-table construction

- Interpreter existence is discharged by a state-transition table per privileged instruction, sized by the restricted state space.
- Valid, because the state space is finite. But it exhibits an object nobody could build for a machine anyone would want.
- It says nothing about whether a correct interpreter is discoverable from the architecture's documentation — the actual engineering problem.
- The paper's own phrase for the hard cases: instructions “of an extremely arcane nature which are in fact arbitrary algorithms.”

The scope of “third generation”

ASSUMED

One boundary, one register

Two modes and a single privilege boundary. One relocation-bounds register, one linear address space, one processor. Interaction only through the program status word.

ARRIVED

Rings, tables, many cores

Multiple privilege rings. Segmentation and paging with structures the guest can read. Multiprocessors observing each other through shared memory. Memory-mapped device registers. System-call instructions that are neither traps nor ordinary transfers.

- The core insight survives — it is still a claim about which state-changing operations are interceptable. The specific test does not transfer mechanically.

08

The theorem after 1974

17

Instructions in the Intel Pentium that are sensitive but not privileged, by Robin and Irvine's 2000 audit against Popek and Goldberg's criteria.

Twenty-six years on, the paper was still the checklist.

Which instructions, and why

INSTRUCTION	WHAT IT DOES IN USER MODE	CLASS IN THE 1974 TAXONOMY
POPF	Silently ignores the interrupt-enable field instead of trapping	Control sensitive · a user mode NOP
SGDT, SIDT	Reads the location of privileged descriptor tables	Behavior sensitive · location
PUSH CS	Exposes the current privilege level in the pushed selector	Behavior sensitive · mode
SMSW	Reads machine status into a general register unprivileged	Behavior sensitive · mode

- POPF is the exact case the paper named in 1974 — “user mode NOP”. The taxonomy classifies all four without modification.

Software around the architecture

1999

Binary translation

Scan guest code before it runs and rewrite the offending instructions into sequences that call the monitor, caching translations so the cost amortises. Theorem 3's hybrid monitor, industrialised — translate rather than interpret.

2003

Paravirtualization

Change the guest so it never issues the offending instructions, replacing them with explicit hypercalls. This abandons the identity clause of the 1974 definition: the guest is no longer running on a duplicate, and has to be ported.

- Adams and Agesen's 2006 comparison found mature binary translation often beat the first generation of hardware support — a caution against assuming the hardware answer is the fast one.

VT-x and AMD-V

- Intel shipped VT-x in 2005, AMD its equivalent the following year: not a new ring, but a new orthogonal root and non-root operation.
- The guest runs its own supervisor code at its own level, while the transitions that matter exit to the monitor — the containment restored by construction.
- The first generation was slow. Nested paging in 2007 and 2008 is what made the hardware path decisively faster.
- The fix was designed against a 1974 criterion, which is why the paper appears throughout the architecture literature of this period.

Nesting in production

- The hardware provides one level of root and non-root operation, not a stack — so the outer monitor must multiplex the single facility among inner ones.
- Theorem 2 assumed a monitor free of timing dependencies. Nested guests are measurably slower, and can tell.
- So the theorem holds while its hypothesis does not — which shows the timing exclusion was never a technicality.
- It is nonetheless ordinary practice: people run their own hypervisors inside rented virtual machines.

Outside the model

CONTAINERS

Resource control without identity

Both control clauses are satisfied, and efficiency far exceeds any monitor. But what the guest sees is a filtered view of one kernel's interfaces, not a duplicate of the machine. The instruction-set test is simply not the relevant question.

UNIKERNELS

Identity without a guest OS

They run on an ordinary virtual machine and change what the guest is instead, collapsing application and kernel into one image. The theorem holds; the object it was written about has gone.

- The test answers only the question the 1974 definition asks. The most consequential isolation mechanisms of the last fifteen years lie outside its scope — which suggests it is the definition, not the theorem, that has aged.

Designing to the theorem

ARM

A dedicated exception level

Virtualization extensions in the v7 architecture, then EL2 as a first-class exception level in v8 — a hypervisor level present in the architecture from the start, rather than a mode retrofitted around leaking instructions.

RISC-V

A specified extension

The hypervisor extension is part of the privileged architecture, ratified in 2021. Virtualizability is treated as a constraint on the instruction set rather than a property to be audited after the fact.

- No new general-purpose architecture since the mid-2000s has shipped failing this test. The condition became a checklist item, then became invisible by being satisfied.

Fifty years, by milestone

1974	1999	2003	2005	2010	2021
■	■	■	■	■	■
Criterion	Translation	Paravirtual	Hardware	Nesting	Specified
the paper	rewrite the code	port the guest	change the chip	Theorem 2, live	RISC-V H

- Thirty-one years between the criterion and hardware satisfying it on the dominant architecture, filled by software. Then fifteen years in which designing to it is simply what architects do.

What the paper established

- A vocabulary — privileged, control sensitive, behavior sensitive, innocuous — still adequate for architectures the authors never saw.
 - A decidable test in the easy direction, cheap enough to run against a principles-of-operation manual.
 - A design requirement, which turned out to be the use that outlasted the others.
 - A demonstration that architectural questions of this kind admit proof rather than only experience.
-
- The paper's own claim is the modest one: previous authors had speculated about the characteristics required, and the formal approach established more precisely the mechanisms to be used and the requirements to be met.

Questions for the room

- Is “an efficient, isolated duplicate of the real machine” still the definition worth proving things about?
- What would a useful necessary condition look like, given that every known escape depends on intended usage?
- Is the claim that I/O requirements are “of a very similar nature” defensible, or the paper's one real evasion?
- If x86 had satisfied the containment in 1985, does the last twenty-five years of systems software look different?
- What is the equivalent unpaid debt in the isolation mechanisms we build today?

09

Questions for you

On the model

- The machine is modelled as a state $\langle E, M, P, R \rangle$ with no general-purpose registers and two privilege modes. What does that abstraction buy the proofs, and which steps would break if registers and multiple rings were modelled?
- Why is the trap defined as saving the *whole* pre-trap state at location 0 and loading a new state from location 1? Which later proof steps depend on it being exactly that protocol?
- The monitor is defined as a triple — dispatcher, allocator, interpreters. Which of the three does Theorem 1's proof actually construct, and which does it assume?
- Equivalence is defined over halting behaviour alone. Name a real virtualization failure that definition cannot see.

On control sensitivity

DEFINITION

State it precisely

Define control sensitivity in terms of the state components the instruction attempts to change, and say why the quantifier is “there exists a state” rather than “for all states”.

EXAMPLE

Find one in x86

Name an x86 instruction that is control sensitive but does *not* trap in user mode, and say exactly which part of the state it tries to touch and what the monitor consequently never learns.

- The paper calls this failure mode a “user mode NOP” in a single 1974 aside. Why is a silently-dropped write worse for a monitor than a fault?
- In one sentence: what distinguishes control sensitivity from behavior sensitivity?

On behavior sensitivity

DEFINITION

State it, and split it

Define behavior sensitivity, then distinguish its two species — location sensitive and mode sensitive — and say what each one lets the guest observe that it should not.

EXAMPLES

One x86 case of each

Give an x86 instruction that is location sensitive and one that is mode sensitive. For each, state precisely what the guest reads and why the monitor cannot intercept the read.

- Which of the two is harder to fix in software, and why?
- Could a monitor satisfy equivalence on a machine with behavior-sensitive instructions if the guest never executed them? What does your answer say about the theorem's escapes?

On the theorems

- Theorem 1 requires the sensitive instructions to be a subset of the privileged ones. Argue why that is sufficient, then argue why it is not necessary. What does the gap tell you about x86?
- The proof discharges interpreter existence by building a state-transition table. Why is that legitimate as an existence proof and useless as an implementation strategy?
- State the extra condition Theorem 2 adds, and the one Theorem 3 relaxes. In one sentence each, why is that the right change?
- Theorem 3 buys virtualizability by interpreting every supervisor-mode instruction. What does that cost, and on which workloads does the cost become unacceptable?

On the limits and the aftermath

- The model assumes a single relocation-bounds memory. Which results survive under multi-level page tables and an IOMMU, and which need restating?
- The paper says nothing about timing, caches or counters. Give a concrete way a modern guest detects virtualization while all three properties hold.
- Rank binary translation, paravirtualization and VT-x by which of the three properties each preserves, sacrifices or sidesteps.
- VT-x makes x86 meet the condition by adding a mode, not by changing the instruction set. Is that a fix to the architecture or to the theorem's applicability? Defend it.
- Nested virtualization is Theorem 2 in the field. What does the theorem predict about nesting depth, and why does the hardware cost more than the recursion suggests?

References

- Popek, G. J. and Goldberg, R. P. Formal requirements for virtualizable third generation architectures. Communications of the ACM 17, 7 (July 1974), 412–421.
- Goldberg, R. P. Architectural principles for virtual computer systems. Ph.D. thesis, Harvard University, 1972 — the long version, and where the I/O and interrupt extensions live.
- Galley, S. W. PDP-10 virtual machines. ACM SIGARCH–SIGOPS Workshop on Virtual Computer Systems, 1969.
- Robin, J. S. and Irvine, C. E. Analysis of the Intel Pentium's ability to support a secure virtual machine monitor. USENIX Security, 2000.
- Barham, P. et al. Xen and the art of virtualization. SOSP, 2003.
- Adams, K. and Agesen, O. A comparison of software and hardware techniques for x86 virtualization. ASPLOS, 2006.
- Ben-Yehuda, M. et al. The Turtles project: design and implementation of nested virtualization. OSDI, 2010.

A

Appendix

Lemma 3, in full

Let i be any instruction, t any sequence of length $\leq k$.

By the induction and lemma hypothesis, for any state S :

$$f\ t(S) = t'\ f(S) \text{ (1)}$$

$$i'\ f\ t(S) = i'\ t'\ f(S) \text{ (2)}$$

$$i'\ f(S) = f\ i(S) \text{ (3)}$$

Letting $t(S)$ be S in (3) and combining with the left of (2):

$$f\ i\ t(S) = i'\ t'\ f(S)$$

- The sequence may be any of length $k + 1$, so this is the induction step. The primed operators need not be the same instructions as the unprimed ones — the induction runs over a correspondence, not an identity.

Definitions, collected

CLASS	CONDITION	QUANTIFIER
Privileged	Traps from user mode, runs from supervisor mode, storage and counter alike	For all such pairs
Control sensitive	Changes R or M without going through the memory trap sequence	There exists a state
Behavior sensitive	Visible window or counter differs when the same program is relocated by x	There exist x and states
User sensitive	Control or behavior sensitive from some state with mode u	There exists a state
Supervisor sensitive	Control or behavior sensitive from some state with mode s	There exists a state
Innocuous	Neither control sensitive nor behavior sensitive	By negation

- Sensitive is the union of the two. The user and supervisor subsets are not necessarily disjoint.