Local Operating Systems R&D And Opportunities for Students

COMP9242

2006/S2 Week 14

OVERVIEW

Cool systems stuff happening at:

- UNSW
  - ➜ Gelato@UNSW
  - ➜ Linux scalability and performance on Itanium
- National ICT Australia (NICTA)
  - ➜ Embedded, Real-Time and Operating Systems (ERTOS) Program
  - ➜ world-class research agenda on embedded operating systems
- Open Kernel Labs, Inc
  - ➜ brand-new ERTOS spinout with a global business
  - ➜ microkernels for millions of people
- Opportunities
  - ➜ summer projects
  - ➜ theses
  - ➜ employment

National ICT Australia (NICTA)

- National Centre of Excellence in Information and Communications Technology (ICT)
- Created in 2003 by Australian Government

NATIONAL ICT AUSTRALIA (NICTA)

- National Centre of Excellence in Information and Communications Technology (ICT)
- Created in 2003 by Australian Government
  - ➜ members: UNSW, ANU, NSW and ACT governments
  - ➜ partners: USyd, UMelb, UQ, QUT, Griffith, QLD+Vic governments
  - ➜ locations: Sydney (Kensington, ATP), Canberra, Melbourne, Brisbane
- Aim: change the Australian ICT landscape
  - ➜ conduct world-class research
  - ➜ improve quality of Australian ICT PhDs
  - ➜ commercialise research outputs
  - ➜ achieve real impact
  - ➜ become one of the top-ten ICT research institutions in the world

NICTA Structure

- Presently $\approx$ 300 researchers, 250 PhD students
- Most researchers belong to *Research Programs*
  - ➜ aligned with discipline areas ($\approx$ 5–10 researchers)
  - ➜ ERTOS is one of them (currently largest)
  - ➜ medium- to long-term vision
- *Projects* focused on specific outcomes
  - ➜ collaborative or client-focused
  - ➜ 1–20 people, 1–5 years

NICTA Structure

- Presently ≈ 300 researchers, 250 PhD students
- Most researchers belong to *Research Programs*
  - ➜ aligned with discipline areas (≈ 5–10 researchers)
  - ➜ ERTOS is one of them (currently largest)
  - ➜ medium- to long-term vision
- *Projects* focused on specific outcomes
  - ➜ collaborative or client-focused
  - ➜ 1–20 people, 1–5 years
- *International Science Advisory Group*
  - ➜ J Vuillemin (VP, INRIA), D Rombach (Head, Fraunhofer IESE), R Newton (Dean UCB), R Brooks (Head MIT CSAIL), S Feldman (VP, IBM)
- *International Business Advisory Group*
  - ➜ D Zitzner, (ret exec VP, HP), N Murthy (Chairman, Infosys), C Mudge (Dir , Macq Innov), B Bishop (V. Chairman, SGI), H Killen (MP Hemisphere Capital)

Embedded, Real-Time and Operating Systems — The ERTOS Program

- One of presently 15 Research Programs in NICTA
  - ➜ 6 PhD-qualified researchers (more being recruited)
  - ➜ 6 engineers / research assistants (1 PhD)
  - ➜ 11 PhD, 1 ME students
- Competencies in
  - ➜ operating systems, microkernels
  - ➜ networking
  - ➜ real-time systems
  - ➜ hardware design

ERTOS Agenda

ERTOS Vision

To make highly reliable, safe and secure embedded systems a widely-deployed reality.

ERTOS Mission

To establish ERTOS-developed embedded operating systems as de-facto industry standards.

EMBEDDED SYSTEM





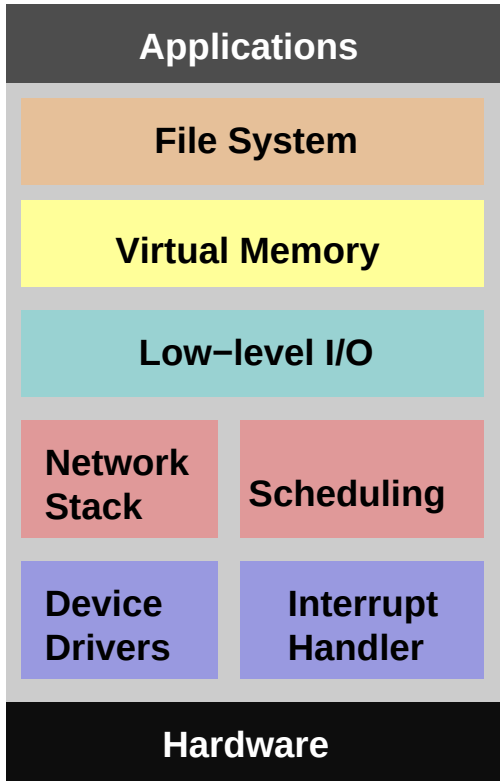



Computer system that is part of a larger system

General-Purpose vs. Embedded

Traditional view:

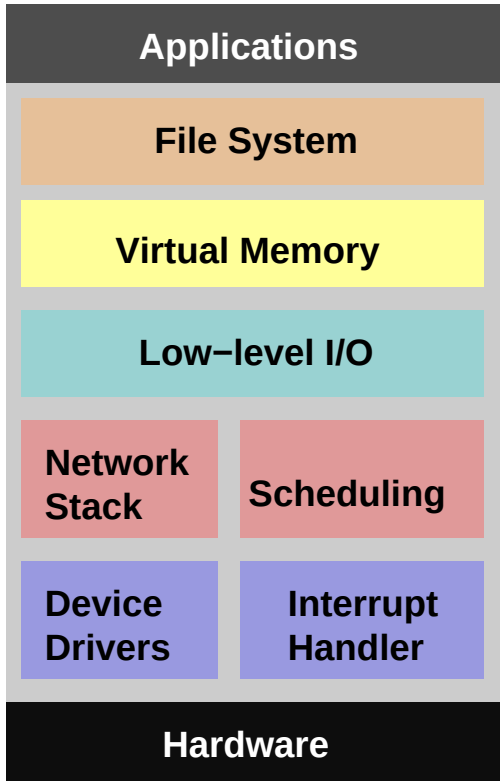
general-purpose system



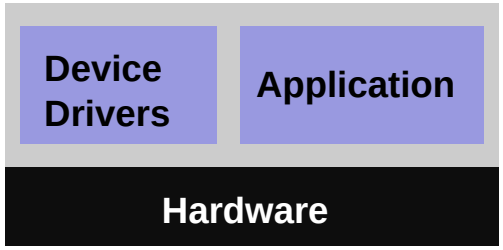
General-Purpose vs. Embedded

Traditional view:

general-purpose system



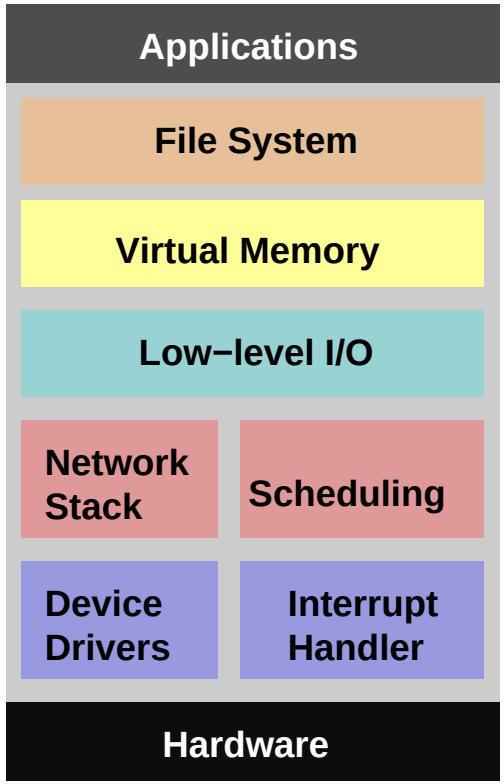
embedded system



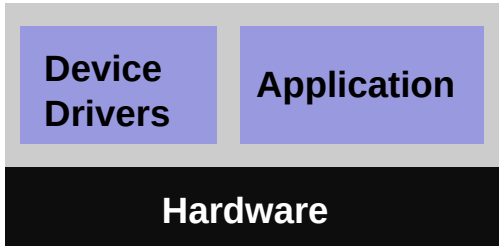
General-Purpose vs. Embedded

Traditional view:

general-purpose system



embedded system



- ➜ minimal
- ➜ no OS at all or small "real-time executive"
- ➜ no protection

Security Challenges

- Growing functionality
- Wireless connectivity
- Downloaded contents (entertainment)
- Increasing dependence on embedded systems

Security Challenges

- Growing functionality
  - ➜ increasing software complexity
    - ➜ millions LOC on phone handsets
    - ➜ Gigabytes of code in cars
  - ➜ increased number of faults
  - ➜ increased likelihood of security faults
- Wireless connectivity
- Downloaded contents (entertainment)
- Increasing dependence on embedded systems

Security Challenges

- Growing functionality
  - ➜ increasing software complexity
    - ➜ millions LOC on phone handsets
    - ➜ Gigabytes of code in cars
  - ➜ increased number of faults
  - ➜ increased likelihood of security faults
- Wireless connectivity
  - ➜ subject to attacks from outside (crackers)
- Downloaded contents (entertainment)
- Increasing dependence on embedded systems

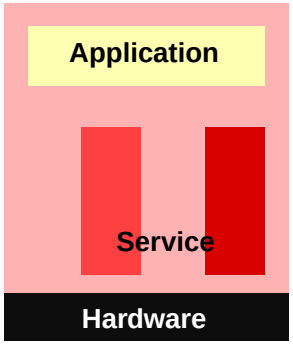
Security Challenges

- Growing functionality
  - ➜ increasing software complexity
    - ➜ millions LOC on phone handsets
    - ➜ Gigabytes of code in cars
  - ➜ increased number of faults
  - ➜ increased likelihood of security faults
- Wireless connectivity
  - ➜ subject to attacks from outside (crackers)
- Downloaded contents (entertainment)
  - ➜ subject to attacks from inside (viruses, worms)
- Increasing dependence on embedded systems

Security Challenges

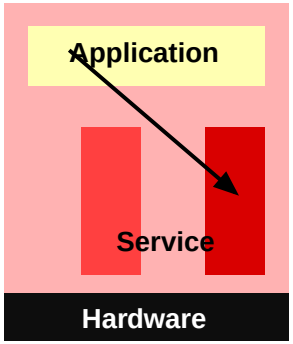
- Growing functionality
  - ➜ increasing software complexity
    - ➜ millions LOC on phone handsets
    - ➜ Gigabytes of code in cars
  - ➜ increased number of faults
  - ➜ increased likelihood of security faults
- Wireless connectivity
  - ➜ subject to attacks from outside (crackers)
- Downloaded contents (entertainment)
  - ➜ subject to attacks from inside (viruses, worms)
- Increasing dependence on embedded systems
  - ➜ increased exposure to embedded-systems security weaknesses

Present approaches 1: Real-time executives



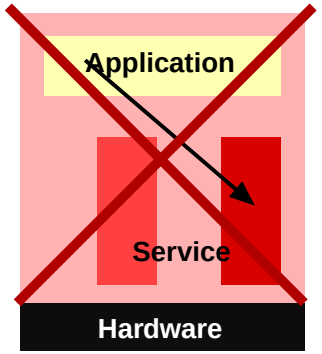
- Small, simple operating system
  - ➜ optimised for fast real-time response
  - ➜ suitable for systems with very limited functionality
- No internal protection

Present approaches 1: Real-time executives



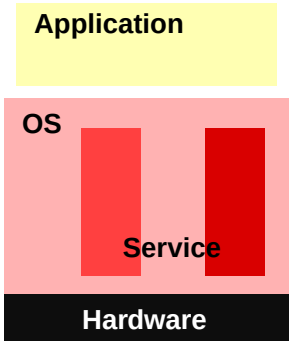
- Small, simple operating system
  - ➜ optimised for fast real-time response
  - ➜ suitable for systems with very limited functionality
- No internal protection

Present approaches 1: Real-time executives



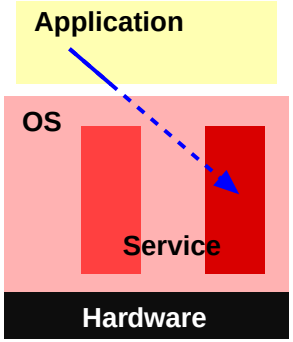
- Small, simple operating system
  - ➜ optimised for fast real-time response
  - ➜ suitable for systems with very limited functionality
- No internal protection
  - ➜ every small bug/failure is fatal
  - ➜ no defence against viruses, limited defence against crackers

Present Approaches 2: Linux, Windows Embedded, ...



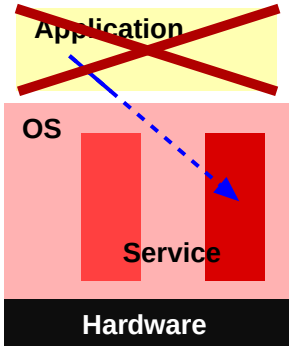
- Scaled-down version of desktop operating system
  ➜ operating system protected from application misbehaviour

Present Approaches 2: Linux, Windows Embedded, ...



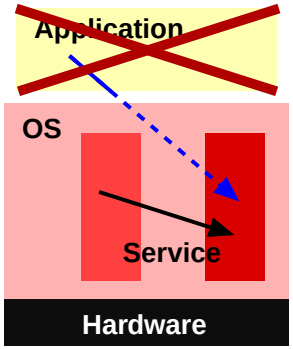
- Scaled-down version of desktop operating system
  ➜ operating system protected from application misbehaviour

Present Approaches 2: Linux, Windows Embedded, ...



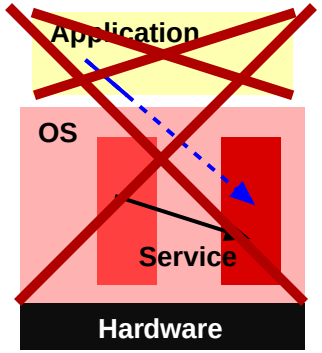
- Scaled-down version of desktop operating system
  ➜ operating system protected from application misbehaviour

Present Approaches 2: Linux, Windows Embedded, ...



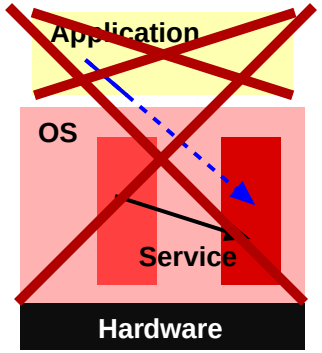
- Scaled-down version of desktop operating system
  ➜ operating system protected from application misbehaviour

Present Approaches 2: Linux, Windows Embedded, ...



- Scaled-down version of desktop operating system
  - ➜ operating system protected from application misbehaviour
  - ➜ excessive code base for small embedded system
  - ➜ too much code on which security of system is dependent

Present Approaches 2: Linux, Windows Embedded, ...



- Scaled-down version of desktop operating system
  - ➜ operating system protected from application misbehaviour
  - ➜ excessive code base for small embedded system
  - ➜ too much code on which security of system is dependent
- Dubious or non-existent real-time capabilities
  - ➜ unsuitable for hard real-time systems

Embedded Systems Requirements:

Reliability, trustworthiness, security:

EMBEDDED SYSTEMS REQUIREMENTS:

Reliability, trustworthiness, security:

- Achieved by:
  - ➜ exhaustive testing?
  - ➜ systematic code inspection?
  - ➜ formal methods?

EMBEDDED SYSTEMS REQUIREMENTS:

Reliability, trustworthiness, security:

- Achieved by:
  - ➜ exhaustive testing?
  - ➜ systematic code inspection?
  - ➜ formal methods?
  - ✗ scale poorly (few 1000 loc)

EMBEDDED SYSTEMS REQUIREMENTS:

Reliability, trustworthiness, security:

- Achieved by:
  - ➜ exhaustive testing?
  - ➜ systematic code inspection?
  - ➜ formal methods?
  - ✗ scale poorly (few 1000 loc)
- Requires minimal *trusted computing base* (TCB):

  TCB: The part of system that must *be relied on for the correct operation of the system*

EMBEDDED SYSTEMS REQUIREMENTS:

Reliability, trustworthiness, security:

- Achieved by:
  - ➜ exhaustive testing?
  - ➜ systematic code inspection?
  - ➜ formal methods?
  - ☒ scale poorly (few 1000 loc)
- Requires minimal *trusted computing base* (TCB):

  TCB: The part of system that must *be relied on for the correct operation of the system*
- Why minimal TCB?
  - ☑ minimise exposure to bugs/faults
  - ☑ minimise exposure to attacks (internal and external)
  - ☑ support poorly-scaling verification methods

Trusted Computing Base

What does the TCB contain?

Trusted Computing Base

What does the TCB contain?

- Kernel
  $\left(\stackrel{\text{def}}{=} \text{part of system that executes in privileged mode}\right)$
  ➜ everything running in privileged mode can bypass security

Trusted Computing Base

What does the TCB contain?

- Kernel
  $\left(\stackrel{\text{def}}{=} \text{part of system that executes in privileged mode}\right)$
  ➜ everything running in privileged mode can bypass security

- Device drivers
  ➜ DMA-capable devices can bypass protection
  ➜ drivers can mount DoS attacks

Trusted Computing Base

What does the TCB contain?

- Kernel
  ($\stackrel{\text{def}}{=}$ part of system that executes in privileged mode)
  ➜ everything running in privileged mode can bypass security

- Device drivers
  ➜ DMA-capable devices can bypass protection
  ➜ drivers can mount DoS attacks

- Services that control resources
  ➜ resource owner can deny resource
  ➜ resource owner can leak/corrupt data

Trusted Computing Base

What does the TCB contain?

- Kernel
  ($\stackrel{\text{def}}{=}$ part of system that executes in privileged mode)
  ➜ everything running in privileged mode can bypass security

- Device drivers
  ➜ DMA-capable devices can bypass protection
  ➜ drivers can mount DoS attacks

- Services that control resources
  ➜ resource owner can deny resource
  ➜ resource owner can leak/corrupt data

- *Everything* on MPU-less processors
  ☒ no memory-protection hardware ⇒ no memory protection

MINIMISING THE SIZE OF THE TCB

... means first of all:

- Use an MPU — microcontrollers are out!

MINIMISING THE SIZE OF THE TCB

... means first of all:

- Use an MPU — microcontrollers are out!
- Minimise the size of the kernel!

MINIMISING THE SIZE OF THE TCB

... means first of all:

- Use an MPU — microcontrollers are out!
- Minimise the size of the kernel!

Minimising kernel size:

- Reduce kernel to what is *essential for supporting secure systems*
- What does not *require* privileged mode *must not* be in the kernel

MINIMISING THE SIZE OF THE TCB

... means first of all:

- Use an MPU — microcontrollers are out!
- Minimise the size of the kernel!

Minimising kernel size:

- Reduce kernel to what is *essential for supporting secure systems*
- What does not *require* privileged mode *must not* be in the kernel
- This is the definition of a *microkernel*

MINIMISING THE SIZE OF THE TCB

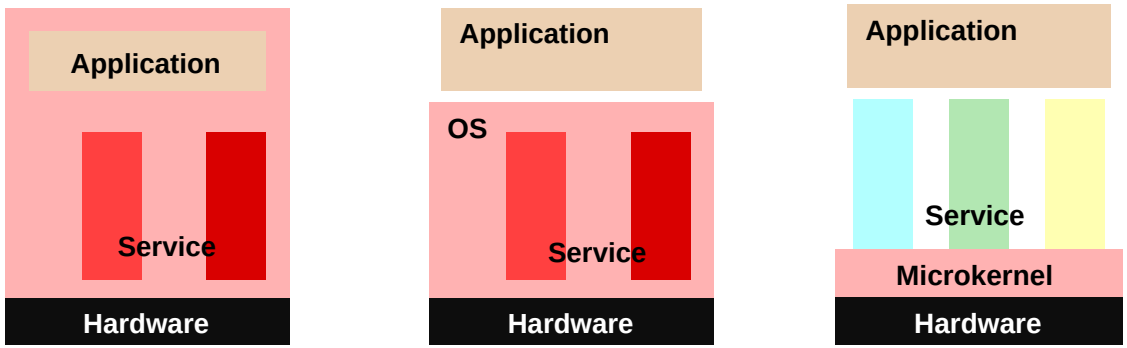
... means first of all:

- Use an MPU — microcontrollers are out!
- Minimise the size of the kernel!

Minimising kernel size:

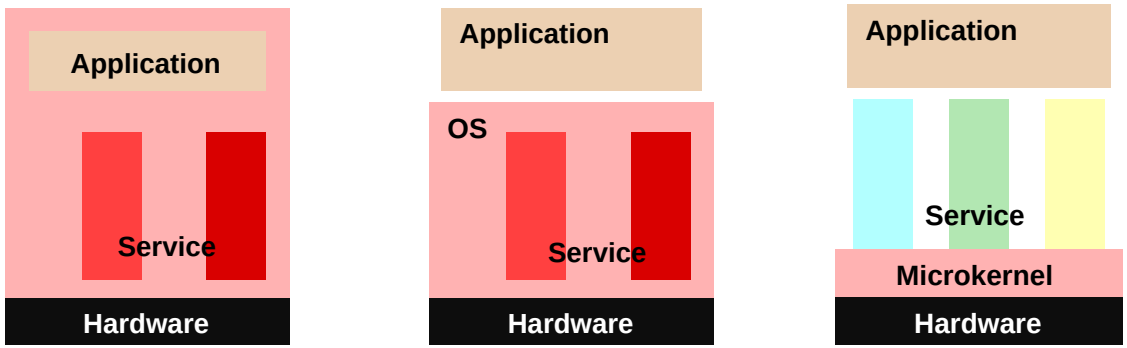
- Reduce kernel to what is *essential for supporting secure systems*
- What does not *require* privileged mode *must not* be in the kernel
- This is the definition of a *microkernel*
- *Minimal TCB required* $\Rightarrow$ *microkernel required!*

Trusted Computing Base



System:	traditional embedded	Linux/ Windows	Microkernel-based
TCB:	**all** code	100,000's loc	10,000's loc

Trusted Computing Base

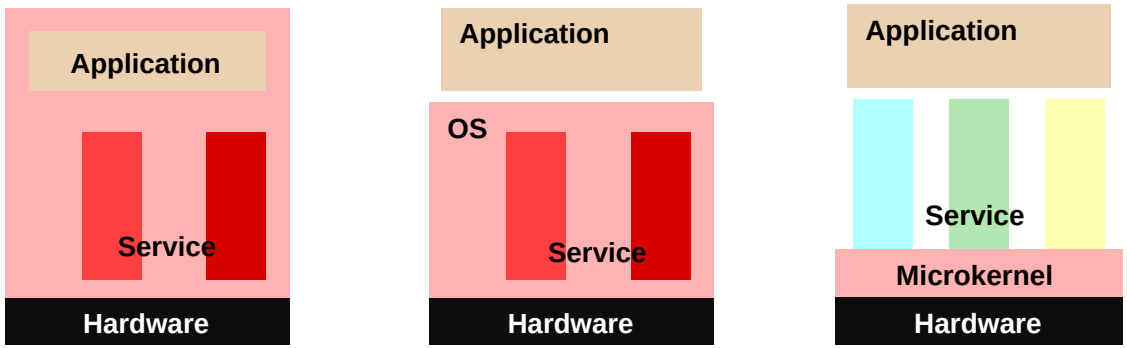


System:	traditional embedded	Linux/ Windows	Microkernel-based
TCB:	**all** code	100,000's loc	10,000's loc

Small is beautiful:

- Small kernel ⇒ potentially small TCB
- Small TCB ⇒ more trustworthy TCB!

TRUSTED COMPUTING BASE



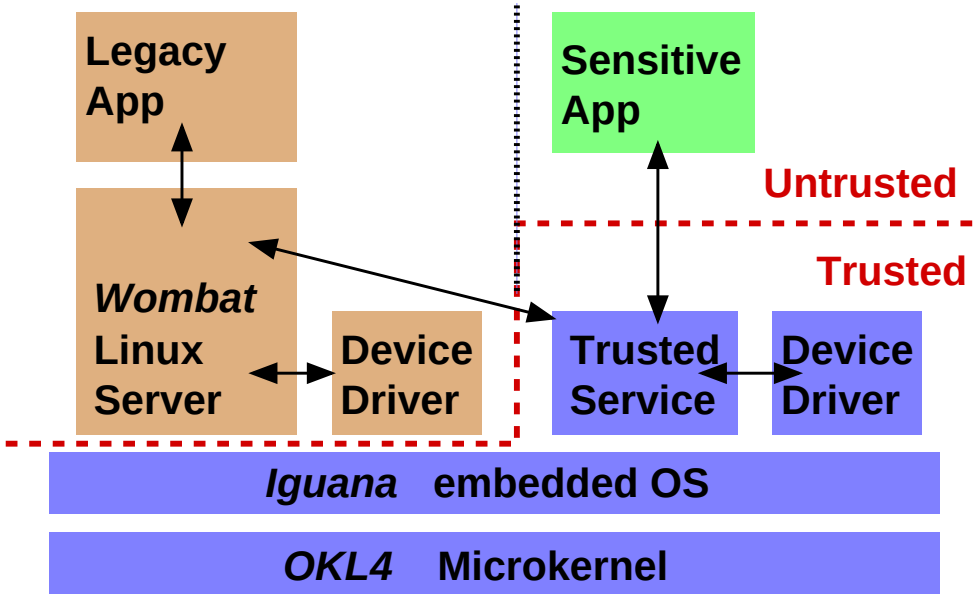
System:	traditional embedded	Linux/ Windows	Microkernel-based
TCB:	**all** code	100,000's loc	10,000's loc

Small is beautiful:

- Small kernel ⇒ potentially small TCB
- Small TCB ⇒ more trustworthy TCB!

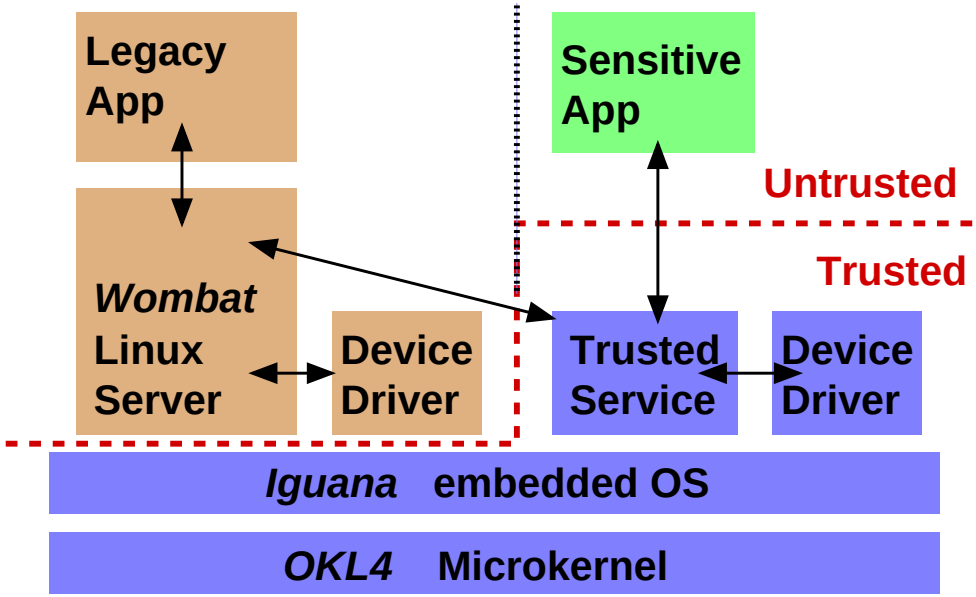
Challenge: Can we *guarantee* the trustworthiness of the TCB?

A Sample System



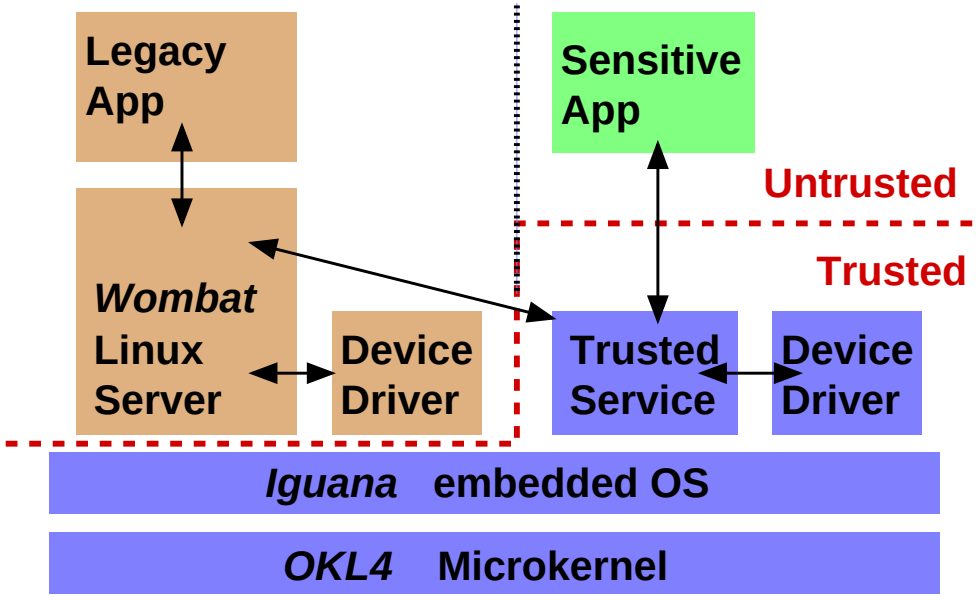
- Sensitive part of system has small TCB

A Sample System



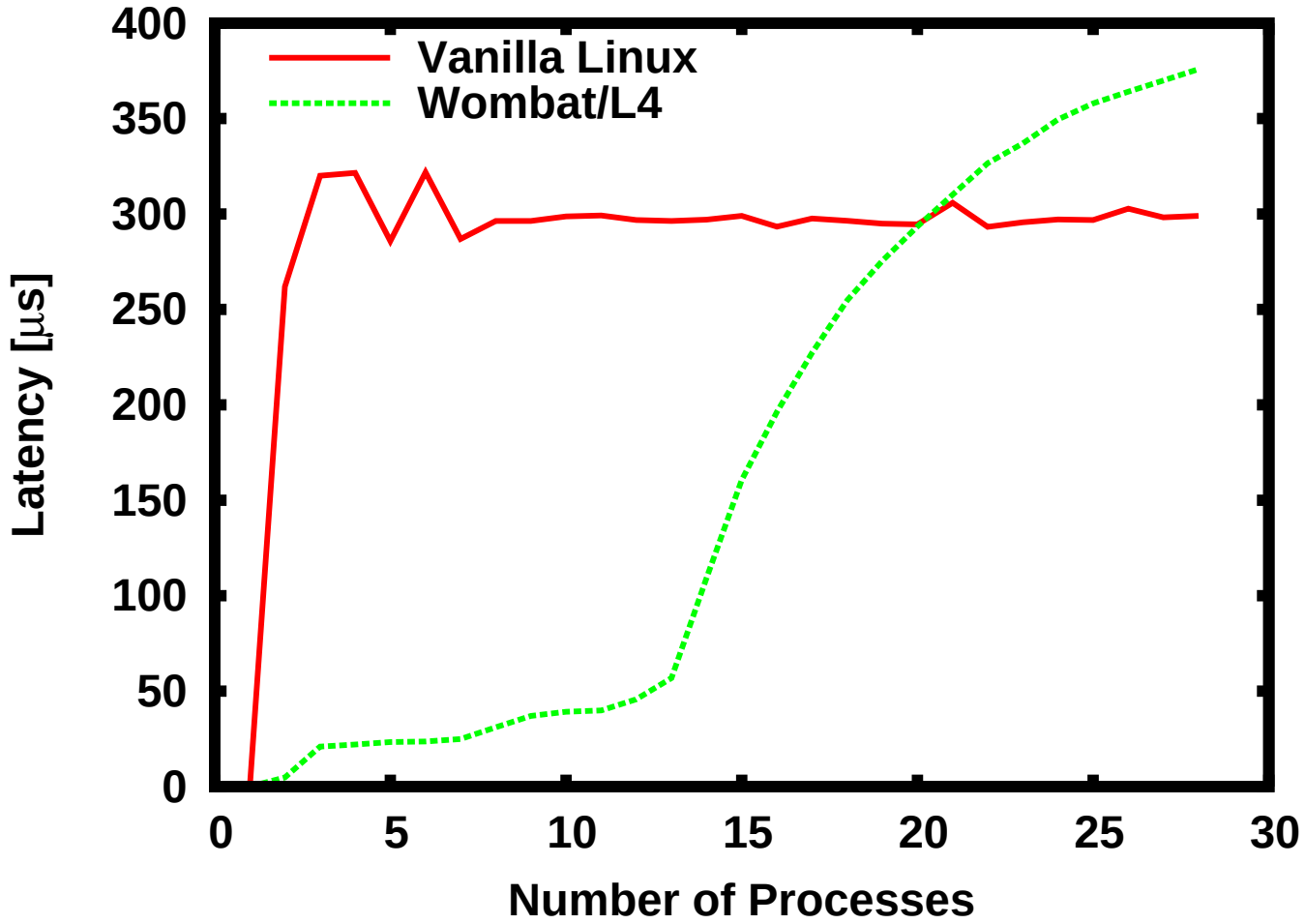
- Sensitive part of system has small TCB
- Standard API supported by de-privileged Linux server
  ➜ full binary compatibility with native Linux

A Sample System

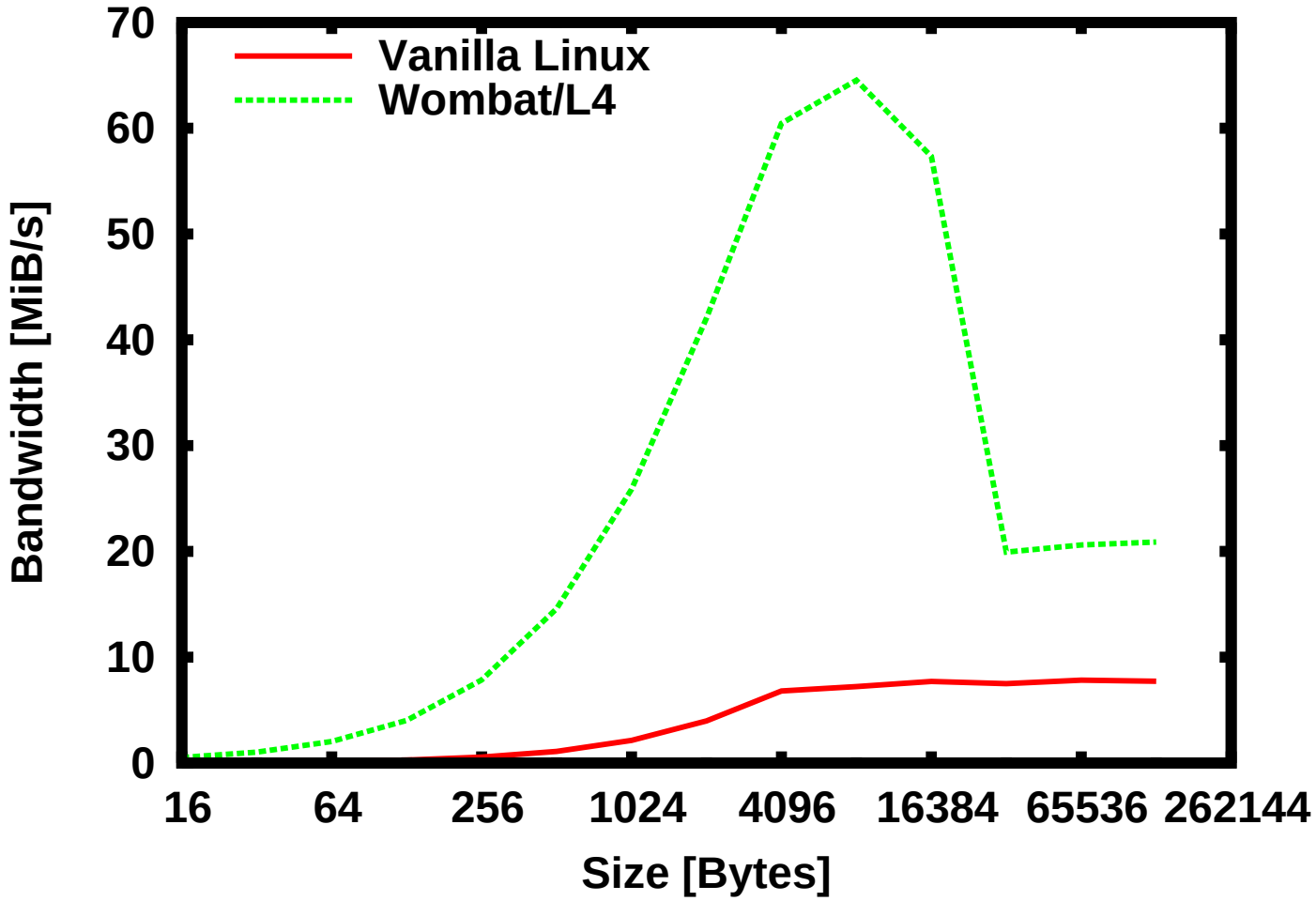


- Sensitive part of system has small TCB
- Standard API supported by de-privileged Linux server
  ➜ full binary compatibility with native Linux
- Compromised legacy system cannot interfere with trusted part

WOMBAT PERFORMANCE: LMBENCH LAT CTX



WOMBAT PERFORMANCE: LMBENCH PIPE



WOMBAT PERFORMANCE: OTHER BENCHMARKS

Benchmark	Linux	Wombat/L4	Ratio
lmbench latencies, (μs), smaller is better			
`lat_fifo`	510	49	10.4
`lat_pipe`	509	49	10.3
`lat_unix`	1015	77	13.3
`lat_sem`	199	14	14.6
AIM7 multitasking benchmark (jobs/min/task)			
1 task	45.15	43.62	0.97
2 tasks	23.35	22.62	0.97
3 tasks	15.79	15.30	0.97

seL4: Microkernel Mechanisms for Secure Systems

seL4: Microkernel for secure embedded systems:

- Security requirements for embedded systems:
  - ➜ Integrity: protecting data from damage
  - ➜ Availability: ensuring system operation
  - ➜ Privacy: protecting sensitive data from loss
  - ➜ IP Protection: controlling propagation of valuable data

seL4: Microkernel Mechanisms for Secure Systems

seL4: Microkernel for secure embedded systems:

- Security requirements for embedded systems:
  - ➜ Integrity: protecting data from damage
  - ➜ Availability: ensuring system operation
  - ➜ Privacy: protecting sensitive data from loss
  - ➜ IP Protection: controlling propagation of valuable data
- Issue: Old L4 API unsuitable for highly-secure systems
  - inefficient information flow control mechanisms
    - ➜ present mechanisms double or triple IPC costs
  - insufficient resource isolation (kernel memory pool)
    - ➜ applications can force kernel to run out of memory
    - ➜ present countermeasures are inflexible

seL4: Microkernel Mechanisms for Secure Systems

seL4: Microkernel for secure embedded systems:

- Security requirements for embedded systems:
  - ➜ Integrity: protecting data from damage
  - ➜ Availability: ensuring system operation
  - ➜ Privacy: protecting sensitive data from loss
  - ➜ IP Protection: controlling propagation of valuable data
- Issue: Old L4 API unsuitable for highly-secure systems
  - inefficient information flow control mechanisms
    - ➜ present mechanisms double or triple IPC costs
  - insufficient resource isolation (kernel memory pool)
    - ➜ applications can force kernel to run out of memory
    - ➜ present countermeasures are inflexible
- Note: Interim solutions for those issues presently in place
  - ➜ seL4 working on clean and general model
  - ➜ production kernel API adapts continuously (and gently)

seL4: Microkernel Mechanisms for Secure Systems

Communications control:

- Targeting confinement, including covert storage channels
- Capability-based IPC authorisation
- Aim: control communication between arbitrary subsystems
- Session-based communications (no *resume cap*)
  - ➜ reply cap passed explicitly on each IPC, or
  - ➜ passed once on session establishment

seL4: Microkernel Mechanisms for Secure Systems

Resource management:

- Aiming at complete control of kernel memory allocation
  - ➜ Verification requires static kernel implementation
  - ➜ Allocation policy dependent on application
  - ➜ Dual systems (Linux + RT) have completing policy requirements

seL4: Microkernel Mechanisms for Secure Systems

Resource management:

- Aiming at complete control of kernel memory allocation
  - ➜ Verification requires static kernel implementation
  - ➜ Allocation policy dependent on application
  - ➜ Dual systems (Linux + RT) have completing policy requirements
- Initially have cap to untyped memory (unused kernel memory)
- Provide model for synthesising new kernel objects (+ caps)

seL4: Microkernel Mechanisms for Secure Systems

Resource management:

- Aiming at complete control of kernel memory allocation
  - ➜ Verification requires static kernel implementation
  - ➜ Allocation policy dependent on application
  - ➜ Dual systems (Linux + RT) have completing policy requirements
- Initially have cap to untyped memory (unused kernel memory)
- Provide model for synthesising new kernel objects (+ caps)
- Possible due to L4 simplicity, small number of objects:
  - ➜ TCBs
  - ➜ Physical frames for virtual memory
  - ➜ synchronous endpoints (like ports with no resume caps)
  - ➜ asynchronous notifation objects
  - ➜ capability nodes
  - ➜ few more for interrupt controllers, page tables, synchronisation

seL4: Microkernel Mechanisms for Secure Systems

Project status:

- Semi-formal API specification in literal Haskell
  - ➜ automatic generation of API documentation from source
  - ➜ draft available from `http://ertos.nicta.com.au/research/sel4`

seL4: Microkernel Mechanisms for Secure Systems

Project status:

- Semi-formal API specification in literal Haskell
  - ➜ automatic generation of API documentation from source
  - ➜ draft available from `http://ertos.nicta.com.au/research/sel4`
- Paper proof of separation properties
  - ➜ suitable for confinement, DRM

seL4: Microkernel Mechanisms for Secure Systems

Project status:

- Semi-formal API specification in literal Haskell
  - ➜ automatic generation of API documentation from source
  - ➜ draft available from `http://ertos.nicta.com.au/research/sel4`
- Paper proof of separation properties
  - ➜ suitable for confinement, DRM
- Prototype implementation in Haskell, integrated with ISA simulator
  - ➜ rapid prototyping: API changes implemented in hours/days
  - ➜ can build and execute apps using standard build tools
  - ➜ used for porting user-level software

seL4: Microkernel Mechanisms for Secure Systems

Project status:

- Semi-formal API specification in literal Haskell
  - ➜ automatic generation of API documentation from source
  - ➜ draft available from `http://ertos.nicta.com.au/research/sel4`
- Paper proof of separation properties
  - ➜ suitable for confinement, DRM
- Prototype implementation in Haskell, integrated with ISA simulator
  - ➜ rapid prototyping: API changes implemented in hours/days
  - ➜ can build and execute apps using standard build tools
  - ➜ used for porting user-level software
- C implementation in progress
  - ➜ prototype in Dec '06

L4.verified: Formal verification of Kernel

- Leverage small size of kernel to *prove* correctness

L4.verified: Formal verification of Kernel

- Leverage small size of kernel to *prove* correctness
- Part 1: Pilot project (Jan '04 – Mar '05)
  - Verified thin "slice" of API all the way to source code
    - ➜ memory-management functions
    - ➜ $>$ 10% of kernel code, $>$ 20% of kernel complexity
    - ➜ 1.5 person years
  - Did almost complete formalisation of present L4 API

L4.verified: Formal verification of Kernel

- Leverage small size of kernel to *prove* correctness
- Part 1: Pilot project (Jan '04 – Mar '05)
  - Verified thin "slice" of API all the way to source code
    - ➜ memory-management functions
    - ➜ $>$ 10% of kernel code, $>$ 20% of kernel complexity
    - ➜ 1.5 person years
  - Did almost complete formalisation of present L4 API
- Part 2: Main project (Apr '05 – Mar '08)
  - Refinement approach using the Isabelle/HOL theorem prover
    - ➜ Importing seL4 API specification (Haskell)
    - ➜ Importing C/assembler implementation to be proved

L4.verified: Formal verification of Kernel

- Leverage small size of kernel to *prove* correctness
- Part 1: Pilot project (Jan '04 – Mar '05)
  - Verified thin "slice" of API all the way to source code
    - ➜ memory-management functions
    - ➜ $>$ 10% of kernel code, $>$ 20% of kernel complexity
    - ➜ 1.5 person years
  - Did almost complete formalisation of present L4 API
- Part 2: Main project (Apr '05 – Mar '08)
  - Refinement approach using the Isabelle/HOL theorem prover
    - ➜ Importing seL4 API specification (Haskell)
    - ➜ Importing C/assembler implementation to be proved
  - Result to be usable in existing deployments
    - ➜ no sacrificing of performance for verifiability
  - On track...

POTOROO: COMPLETE TEMPORAL MODEL OF KERNEL

- Prerequisite for complete real-time analysis of whole system
  - ➜ strict worst-case execution times (WCET)
  - ➜ probabilistic WCET

POTOROO: COMPLETE TEMPORAL MODEL OF KERNEL

- Prerequisite for complete real-time analysis of whole system
  - ➜ strict worst-case execution times (WCET)
  - ➜ probabilistic WCET
- Essential for trustworthy real-time systems
  - ➜ RT analysis of applications pointless without timing model of kernel

POTOROO: COMPLETE TEMPORAL MODEL OF KERNEL

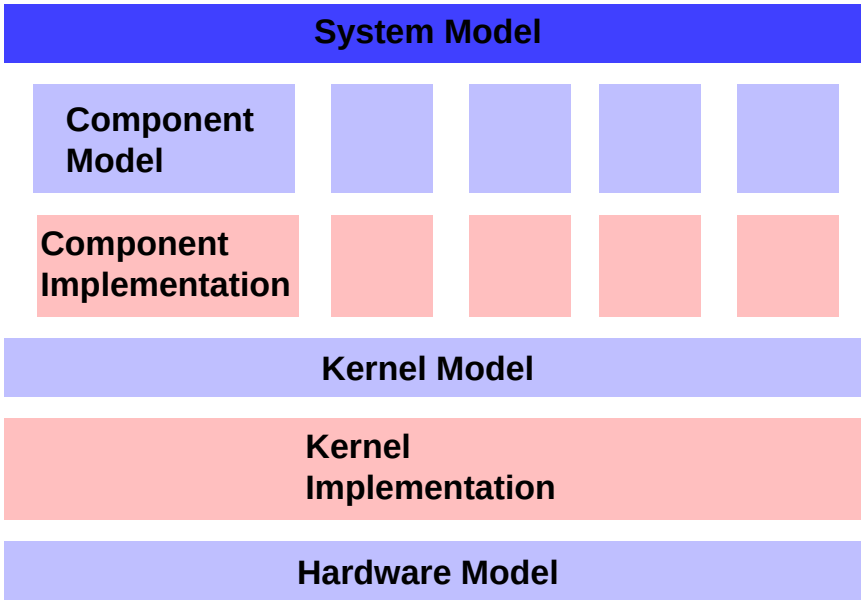
- Prerequisite for complete real-time analysis of whole system
  - ➜ strict worst-case execution times (WCET)
  - ➜ probabilistic WCET
- Essential for trustworthy real-time systems
  - ➜ RT analysis of applications pointless without timing model of kernel
- Measurement-based approach augmented by static analysis
  - ➜ *measure* execution-time profiles of basic blocks
  - ➜ convolute into overall execution-time profile
  - ➜ static analysis to ensure worst case observed
  - ➜ static analysis to reduce pessimism

CAmkES: Component Architecture for Microkernel-Based Embedded Systems

- Aim: approach for highly-componentised embedded software
  - ☑ reduce software cost by enforcing modularity
  - ☑ deliver on fault isolation, hot upgrades, security enforcement, ...

CAmkES: Component Architecture for Microkernel-Based Embedded Systems

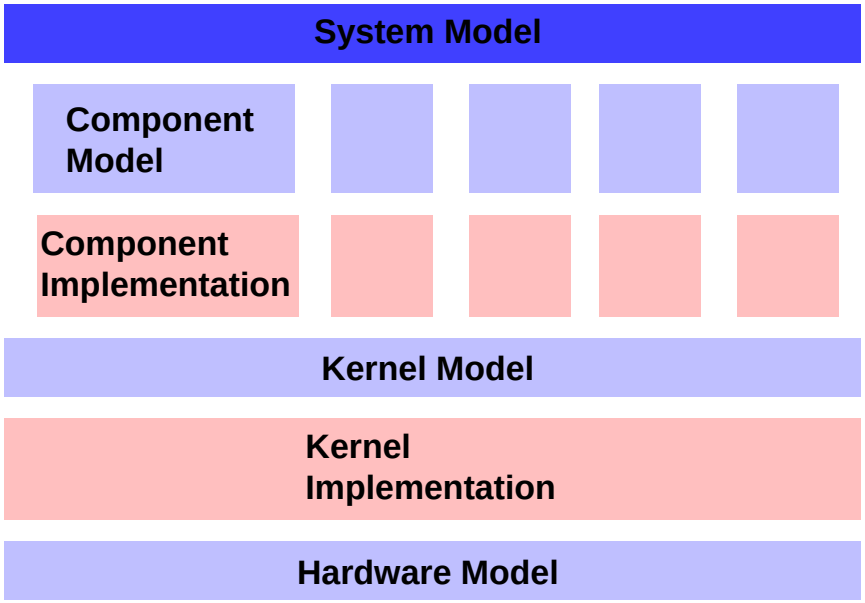
- Aim: approach for highly-componentised embedded software
  - ☑ reduce software cost by enforcing modularity
  - ☑ deliver on fault isolation, hot upgrades, security enforcement, ...
- Ultimate goal:
  Full system verification



CAMKES: COMPONENT ARCHITECTURE FOR MICROKERNEL-BASED EMBEDDED SYSTEMS

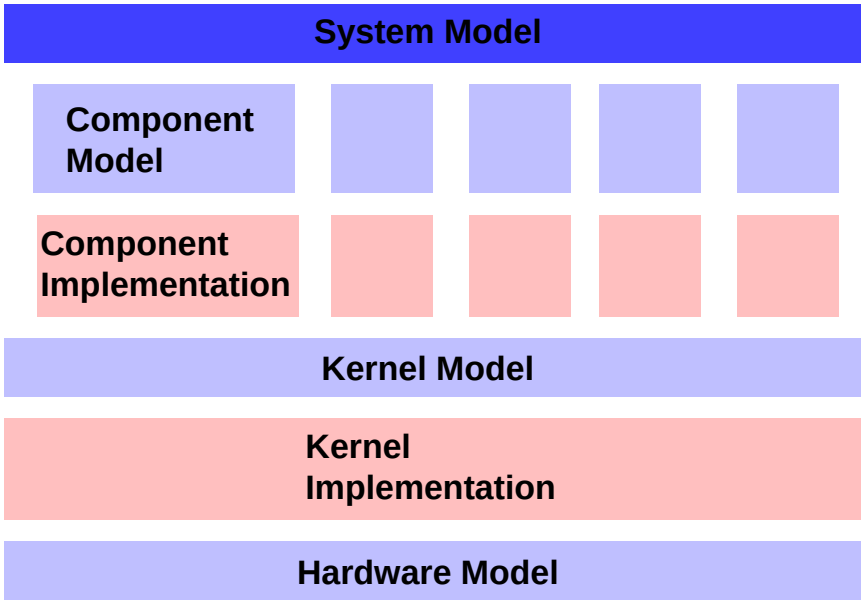
- Aim: approach for highly-componentised embedded software
  - ☑ reduce software cost by enforcing modularity
  - ☑ deliver on fault isolation, hot upgrades, security enforcement, ...
- Ultimate goal:

  Full system verification
  - ➜ kernel-enforced component boundaries
  - ☑ can verify components individually
  - ☒ model composition?
  - ➜ Distant future...



CAMKES: COMPONENT ARCHITECTURE FOR MICROKERNEL-BASED EMBEDDED SYSTEMS

- Aim: approach for highly-componentised embedded software
  - ☑ reduce software cost by enforcing modularity
  - ☑ deliver on fault isolation, hot upgrades, security enforcement, ...
- Ultimate goal:
  Full system verification
  - ➜ kernel-enforced component boundaries
  - ☑ can verify components individually
  - ☒ model composition?
  - ➜ Distant future...
- Status: static prototype



- Working on dynamic system, performance, non-functional properties

HIGH-PERFORMANCE USER-LEVEL DRIVERS

- L4 device drivers are *always* outside the kernel (at user level)
  - Interrupts delivered to driver as IPC from kernel

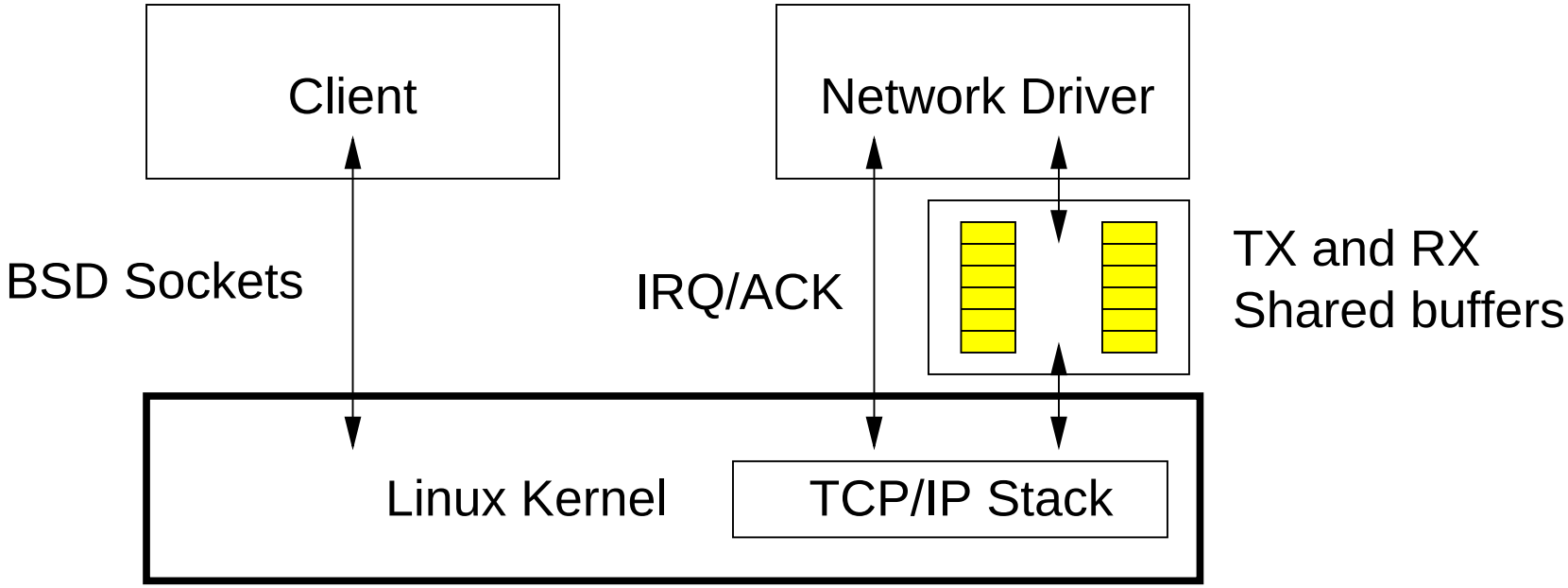
HIGH-PERFORMANCE USER-LEVEL DRIVERS

- L4 device drivers are *always* outside the kernel (at user level)
  - Interrupts delivered to driver as IPC from kernel
- Potentially higher communication overhead
  - past experience with user-level drivers: $\geq$50% performance degradation

HIGH-PERFORMANCE USER-LEVEL DRIVERS

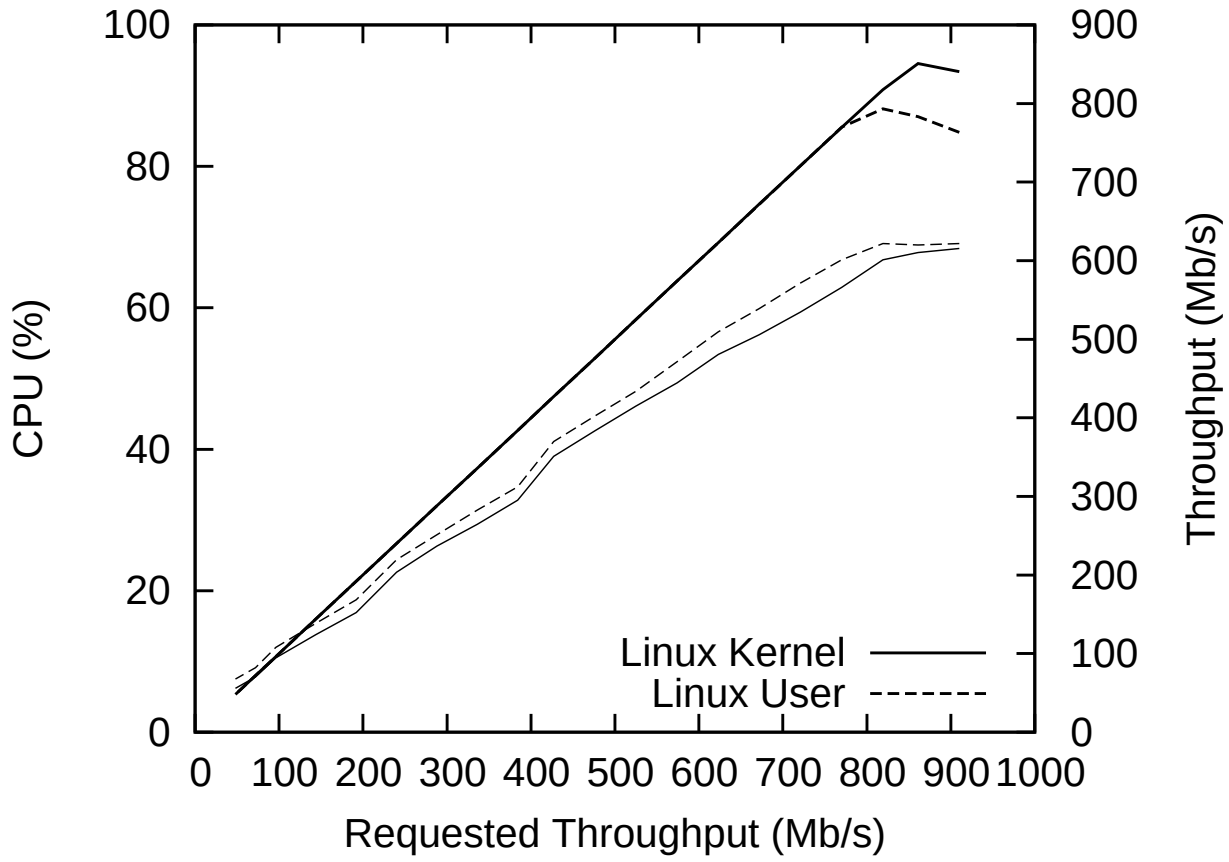
- L4 device drivers are *always* outside the kernel (at user level)
  - Interrupts delivered to driver as IPC from kernel
- Potentially higher communication overhead
  - past experience with user-level drivers: $\geq$50% performance degradation
- L4 IPC performance is very high
  - with well-designed driver interfaces can achieve good performance

User-Level Device Drivers on Linux



Benchmarking setup

User-Level Device Driver Performance



Gigabit Ethernet echo on 900MHz Itanium-2 with 66MHz 64-bit PCI

User-Level Drivers: Ongoing Work

- Complete driver framework and methodology
  - ➜ ease development of high-performance drivers
  - ➜ reduce driver complexity
  - ➜ drivers portable between systems (L4 and Linux)

User-Level Drivers: Ongoing Work

- Complete driver framework and methodology
  - ➜ ease development of high-performance drivers
  - ➜ reduce driver complexity
  - ➜ drivers portable between systems (L4 and Linux)
- Integration with I/O system
  - ➜ Linux VFS layer integration
  - ➜ user-level network protocol stacks
  - ➜ componentised protocol stacks

User-Level Drivers: Ongoing Work

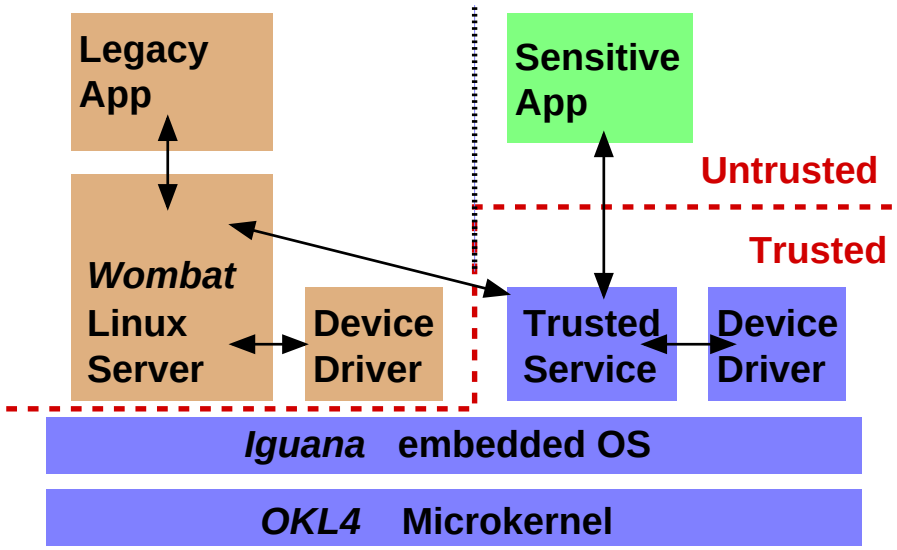
- Complete driver framework and methodology
  - ➜ ease development of high-performance drivers
  - ➜ reduce driver complexity
  - ➜ drivers portable between systems (L4 and Linux)
- Integration with I/O system
  - ➜ Linux VFS layer integration
  - ➜ user-level network protocol stacks
  - ➜ componentised protocol stacks
- Driver encapsulation
  - ➜ use hardware mechanisms to limit DMA
  - ➜ use software mechanisms to limit trust in drivers
  - ➜ goal: untrusted device drivers

User-Level Drivers: Ongoing Work

- Complete driver framework and methodology
  - ➜ ease development of high-performance drivers
  - ➜ reduce driver complexity
  - ➜ drivers portable between systems (L4 and Linux)
- Integration with I/O system
  - ➜ Linux VFS layer integration
  - ➜ user-level network protocol stacks
  - ➜ componentised protocol stacks
- Driver encapsulation
  - ➜ use hardware mechanisms to limit DMA
  - ➜ use software mechanisms to limit trust in drivers
  - ➜ goal: untrusted device drivers
- Collaboration between NICTA and UNSW Gelato project
  - ➜ 1 PhD student

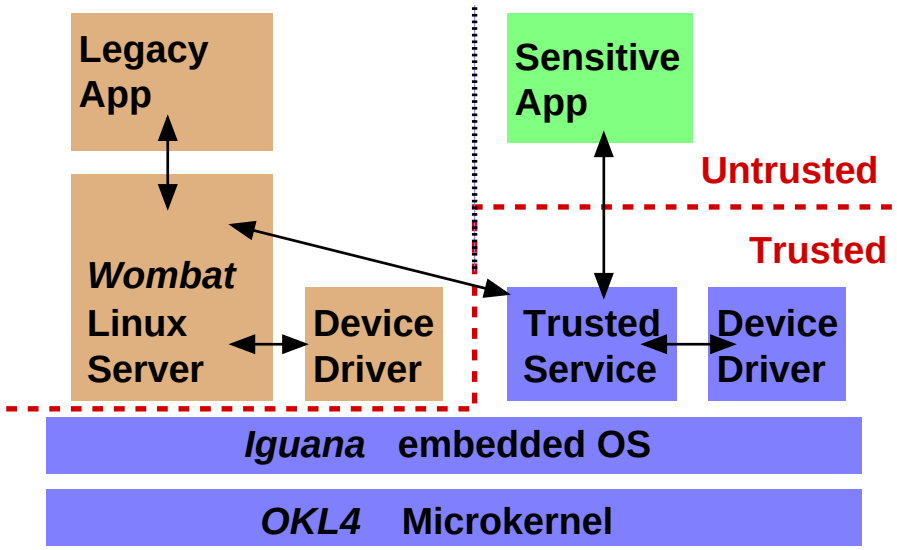
PRESENT STATE

- Pistachio: Mature microkernel
  - ➜ 10,000 lines of code (shrinking)
  - ➜ highly efficient
- Iguana: Core OS services
  - ➜ naming, protection, memory...
  - ➜ device drivers
  - ➜ optional Linux server



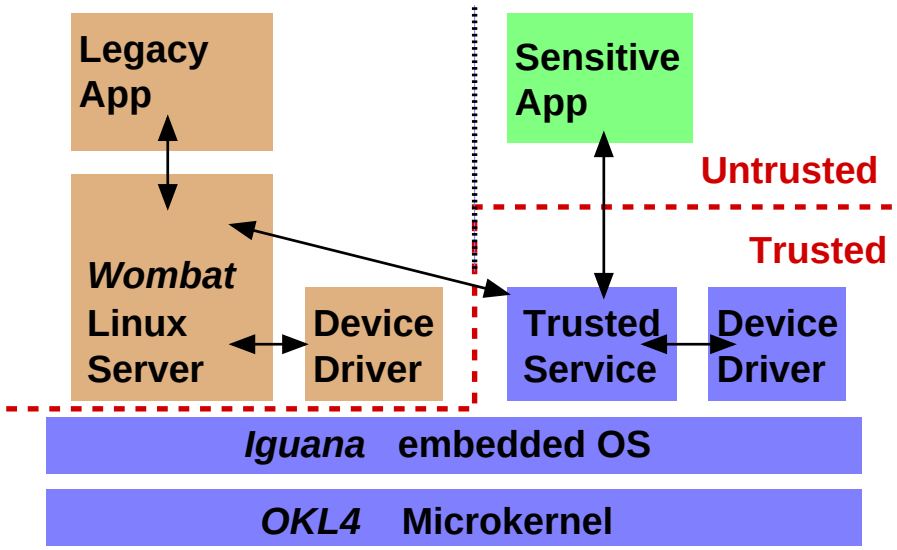
PRESENT STATE

- Pistachio: Mature microkernel
  - ➜ 10,000 lines of code (shrinking)
  - ➜ highly efficient
- Iguana: Core OS services
  - ➜ naming, protection, memory...
  - ➜ device drivers
  - ➜ optional Linux server
- Multiple architectures
  - ➜ on ARM, MIPS, x86



PRESENT STATE

- Pistachio: Mature microkernel
  - ➜ 10,000 lines of code (shrinking)
  - ➜ highly efficient
- Iguana: Core OS services
  - ➜ naming, protection, memory...
  - ➜ device drivers
  - ➜ optional Linux server
- Multiple architectures
  - ➜ on ARM, MIPS, x86
- Commercially deployed
  - ➜ new base of Qualcomm CDMA chip firmware
  - ➜ other deployments in pipeline



Open Kernel Labs (OKL)

- Startup company for commercialising ERTOS technology
- Created Sep 2006 (still in setup process)
  - Steve Subar, CEO
    ➜ startup veteran
  - Gernot Heiser, CTO
  - ca 15 engineers, growing 1-2 per month
    ➜ probably largest group of top kernel hackers outside major multinationals
  - US HQ, Sydney-based engineering
- Projects with 5 large multinationals, several others in pipeline
  ➜ mobile communication chipsets and phone handsets
  ➜ multimedia
  ➜ some huge stuff we can't talk about

Open Kernel Labs — A Unique Approach

OKL-NICTA Joint Venture:

- OKL provides services
- NICTA/ERTOS does research
- Outcomes industrialised and commercialised by OKL

OKL/NICTA Ongoing Relationship:

- Students move into either OKL or ERTOS
  ➜ working on similar stuff
- ERTOS staff move into OKL with their projects
  ➜ efficient industrialisation/commercialisation
- OKL staff move back to ERTOS
  ➜ do PhD on research issues identified by OKL

Student Opportunities

Would you like to work on cool systems people actually use???

- Gelato — kernel work for supercomputers
  ➜ with significant research issues
- BLUEsat — L4 in space!
  ➜ but first it needs an OS!
- ERTOS research — trustworthy embedded systems
  ➜ ... will change the industry!
- Open Kernel Labs — microkernels in billions of devices
  ➜ hot startup building cool systems