Operating Systems Research at UNSW and NICTA

and

Opportunities for Students

COMP9242 2004/S2 Week 14

OVERVIEW

- What is NICTA?
- Embedded systems research agenda
- Present projects
- Opportunities

National ICT Australia (NICTA)

- National Centre of Excellence in Information and Communications Technology (ICT)
- Created in October 2002 by Australian Government

National ICT Australia (NICTA)

- National Centre of Excellence in Information and Communications Technology (ICT)

- Created in October 2002 by Australian Government

- Members:
  - University of New South Wales (UNSW), Sydney
  - Australian National University (ANU), Canberra
  - NSW and ACT governments

NATIONAL ICT AUSTRALIA (NICTA)

- National Centre of Excellence in Information and Communications Technology (ICT)

- Created in October 2002 by Australian Government

- Members:
  - ⋆ University of New South Wales (UNSW), Sydney
  - ⋆ Australian National University (ANU), Canberra
  - ⋆ NSW and ACT governments

- Locations:
  - ⋆ Sydney (UNSW campus and ATP technology park)
  - ⋆ Canberra
  - ⋆ Melbourne (University of Melbourne Campus)
  - ⋆ Brisbane (in progress)

NICTA's Four Pillars

- Research
- Education
- Linkages
- Commercialisation

NICTA's Four Pillars

- Research
  - ➜ conduct world-class ICT research that makes an impact
  - ➜ "use-inspired fundamental research"
- Education
- Linkages
- Commercialisation

NICTA's Four Pillars

- Research
  - ➜ conduct world-class ICT research that makes an impact
  - ➜ "use-inspired fundamental research"
- Education
  - ➜ produce world-class PhD graduates
  - ➜ students enrolled at one of the member/partner universities
- Linkages
- Commercialisation

NICTA's Four Pillars

- Research
  - ➜ conduct world-class ICT research that makes an impact
  - ➜ "use-inspired fundamental research"
- Education
  - ➜ produce world-class PhD graduates
  - ➜ students enrolled at one of the member/partner universities
- Linkages
  - ➜ collaborate with top research institutions around the world
- Commercialisation

NICTA's Four Pillars

- Research
  - ➜ conduct world-class ICT research that makes an impact
  - ➜ "use-inspired fundamental research"
- Education
  - ➜ produce world-class PhD graduates
  - ➜ students enrolled at one of the member/partner universities
- Linkages
  - ➜ collaborate with top research institutions around the world
- Commercialisation
  - ➜ turn research into products
  - ➜ focus on
    - ➜ local small and medium-sized enterprises (SMEs)
    - ➜ multinational corporations (MNCs)

NICTA Structure

- Presently $\approx$ 150 researchers, 150 PhD students

NICTA STRUCTURE

- Presently ≈ 150 researchers, 150 PhD students
- Researchers belong to *Research Programs*
  - ➜ aligned with discipline areas (≈ 5–10 researchers)
  - ➜ lifetime 5–10 years

NICTA Structure

- Presently ≈ 150 researchers, 150 PhD students
- Researchers belong to *Research Programs*
  - ➜ aligned with discipline areas (≈ 5–10 researchers)
  - ➜ lifetime 5–10 years
- *Priority Challenges* drive research
  - ➜ PC1: trusted wireless networks
  - ➜ PC2: from data to knowledge

NICTA STRUCTURE

- Presently ≈ 150 researchers, 150 PhD students
- Researchers belong to *Research Programs*
  - ➜ aligned with discipline areas (≈ 5–10 researchers)
  - ➜ lifetime 5–10 years
- *Priority Challenges* drive research
  - ➜ PC1: trusted wireless networks
  - ➜ PC2: from data to knowledge
- *Projects* focused on specific outcomes
  - ➜ collaborative or client-focused
  - ➜ lifetime 1–5 years

NICTA STRUCTURE

- Presently ≈ 150 researchers, 150 PhD students
- Researchers belong to *Research Programs*
  - ➜ aligned with discipline areas (≈ 5–10 researchers)
  - ➜ lifetime 5–10 years
- *Priority Challenges* drive research
  - ➜ PC1: trusted wireless networks
  - ➜ PC2: from data to knowledge
- *Projects* focused on specific outcomes
  - ➜ collaborative or client-focused
  - ➜ lifetime 1–5 years
- *International Science Advisory Group*
  - ➜ Richard Newton (UCB), Shankar Sastry (UCB), Raj Reddy (CMU), Rodney Brooks (MIT), Jeff Ullman (Stanford), Gunnar Bjurel (SIKS), Gilles Kahn (INRIA), Ya-Qin. Zhang (Microsoft), ...

EMBEDDED, REAL-TIME AND OPERATING SYSTEMS (ERTOS) PROGRAM

- One of presently 16 Research Programs in NICTA
  - ➜ 8 FTE PhD-qualified researchers (1 vacant)
  - ➜ 11 FTE research engineers / research assistants (2 PhDs)
  - ➜ 20 PhD students (10 core OS topics)
- Competencies in
  - ➜ operating systems, microkernels
  - ➜ networking
  - ➜ real-time systems
  - ➜ reconfigurable computing
  - ➜ programming languages and compiler front-ends

Embedded Systems in Australia

- Australian embedded systems industry landscape
  - ⋆ little industrial research
  - ⋆ innovation concentrated in SMEs
  - ⋆ little confidence in locally-developed technology
  - ⋆ operating in niche markets

Embedded Systems in Australia

- Australian embedded systems industry landscape
  - ⋆ little industrial research
  - ⋆ innovation concentrated in SMEs
  - ⋆ little confidence in locally-developed technology
  - ⋆ operating in niche markets
- Implications:
  - ⋆ no scope for ASICs, use COTS hardware components
  - ⋆ main focus is on software

EMBEDDED SYSTEMS IN AUSTRALIA

- Australian embedded systems industry landscape
  - ⋆ little industrial research
  - ⋆ innovation concentrated in SMEs
  - ⋆ little confidence in locally-developed technology
  - ⋆ operating in niche markets
- Implications:
  - ⋆ no scope for ASICs, use COTS hardware components
  - ⋆ main focus is on software
  - ⋆ reconfigurable hardware will be increasingly important
    - ➜ prototyping
    - ➜ small series
    - ➜ flexibility

NICTA Priority Challenge:
Trusted Wireless Networks

To enable greater confidence, freedom, and capability through improved efficiency, reliability, and security of all wireless environments.

NICTA Priority Challenge: Trusted Wireless Networks

To enable greater confidence, freedom, and capability through improved efficiency, reliability, and security of all wireless environments.

- Strongly based on embedded systems technology

NICTA PRIORITY CHALLENGE: TRUSTED WIRELESS NETWORKS

To enable greater confidence, freedom, and capability through improved efficiency, reliability, and security of all wireless environments.

- Strongly based on embedded systems technology
- Issues relevant to ERTOS:
  - ⋆ efficiency
  - ⋆ reliability
  - ⋆ security
  - ⋆ cost

ERTOS VISION

To develop methodologies, tools, components and systems that will deliver reliable, trustworthy and inexpensive embedded systems software

ERTOS VISION

To develop methodologies, tools, components and systems that will deliver reliable, trustworthy and inexpensive embedded systems software

ERTOS research is to be driven by applications

- to identify common challenges
- to provide generic systems software

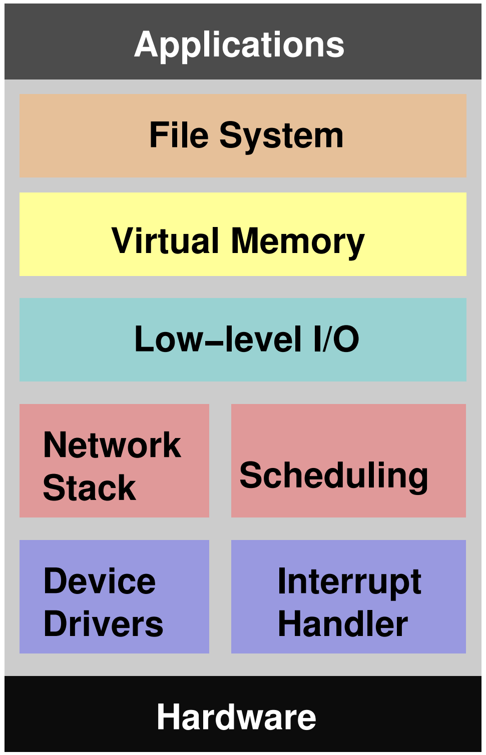
EMBEDDED SYSTEM

Computer system that is part of a larger system

General-Purpose vs. Embedded

Traditional view:

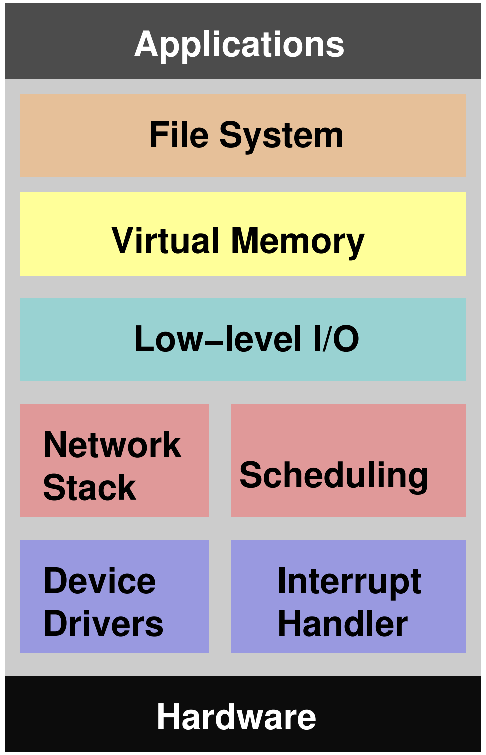
general-purpose system



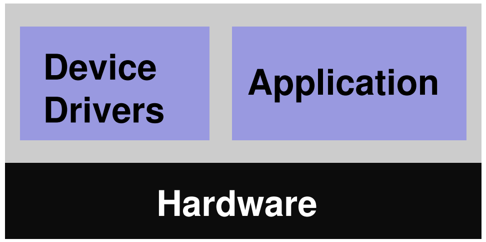
General-Purpose vs. Embedded

Traditional view:

general-purpose system



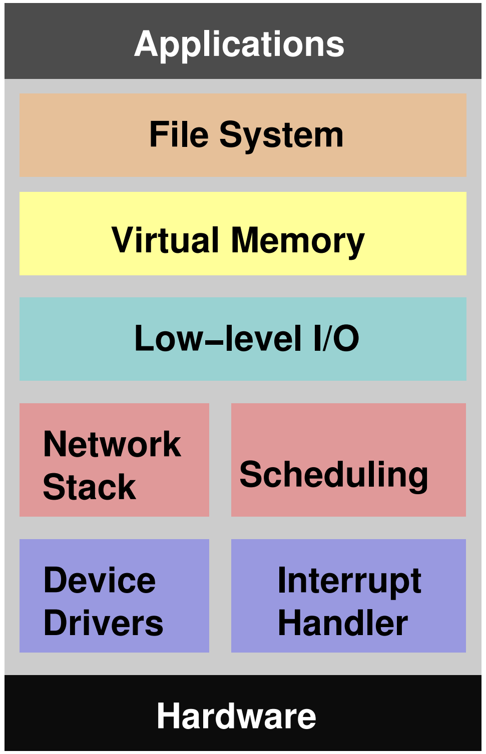
embedded system



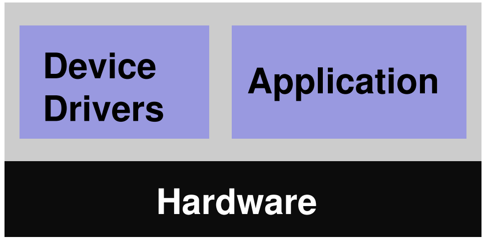
GENERAL-PURPOSE VS. EMBEDDED

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)

Security Challenges

- Growing functionality

  ➜ increasing software complexity
  ➜ increased number of faults
  ➜ increased likelihood of security faults

- Wireless connectivity

- Downloaded contents (entertainment)

Security Challenges

- Growing functionality
  - ➜ increasing software complexity
  - ➜ increased number of faults
  - ➜ increased likelihood of security faults
- Wireless connectivity
  - ➜ subject to attacks from outside (crackers)
- Downloaded contents (entertainment)

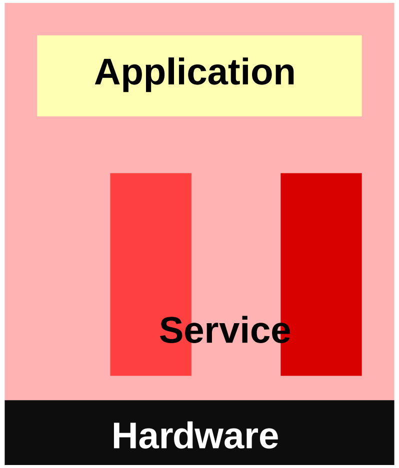
Security Challenges

- Growing functionality
  - ➜ increasing software complexity
  - ➜ 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)

SECURITY CHALLENGES

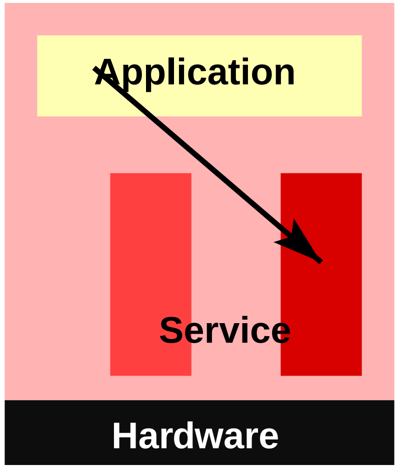
- Growing functionality
  - ➜ increasing software complexity
  - ➜ increased number of faults
  - ➜ increased likelihood of security faults
- Wireless connectivity
  - ➜ subject to attacks from outside (crackers)
- Downloaded contents (entertainment)
  - ➜ subject to attacks from inside (viruses, worms)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

PRESENT APPROACHES 1: REAL-TIME EXECUTIVES



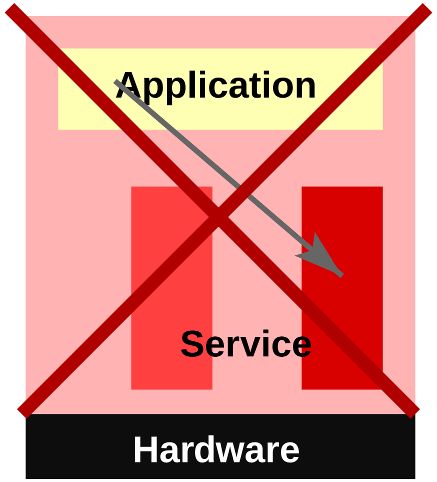
- Small, simple operating system
  - ➜ optimised for fast real-time response
  - ➜ suitable for systems with very limited functionality
- No internal protection

PRESENT APPROACHES 1: REAL-TIME EXECUTIVES



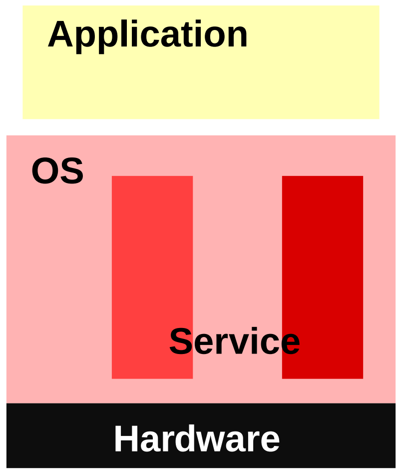
- Small, simple operating system
  - ➜ optimised for fast real-time response
  - ➜ suitable for systems with very limited functionality
- No internal protection

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

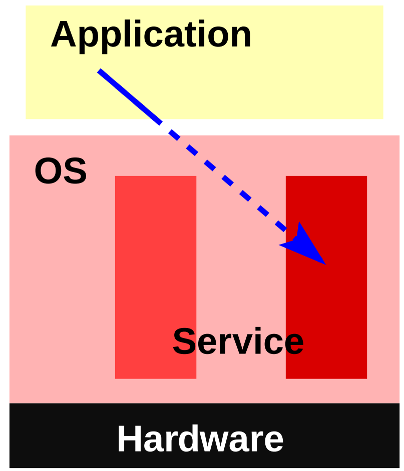
PRESENT APPROACHES 2: LINUX, WINDOWS EMBEDDED, ...



- Scaled-down version of desktop operating system

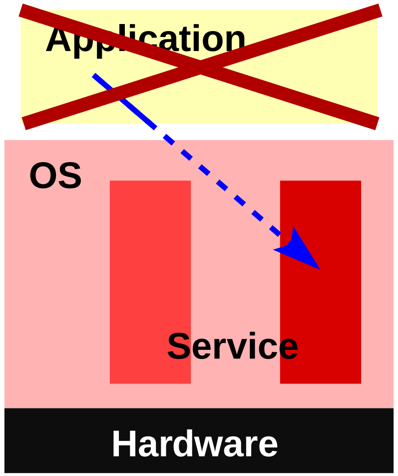
➜ operating system protected from application misbehaviour

PRESENT APPROACHES 2: LINUX, WINDOWS EMBEDDED, ...



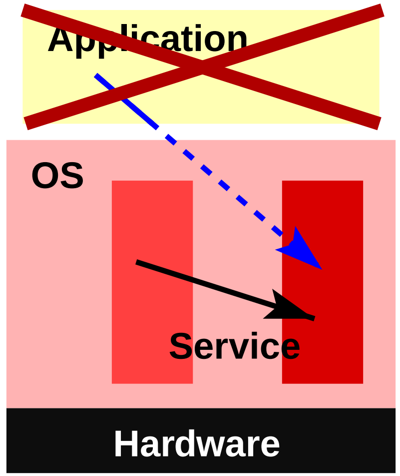
- Scaled-down version of desktop operating system
  - ➜ operating system protected from application misbehaviour

PRESENT APPROACHES 2: LINUX, WINDOWS EMBEDDED, ...



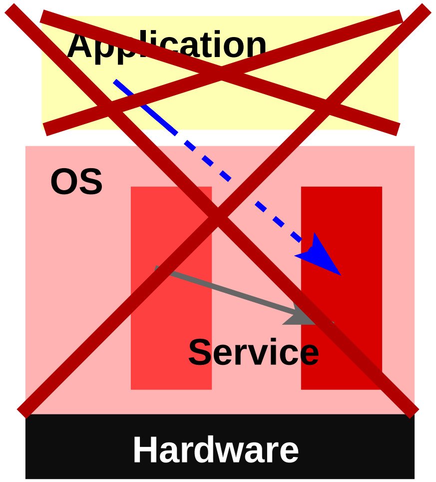
- Scaled-down version of desktop operating system
  - ➜ operating system protected from application misbehaviour

PRESENT APPROACHES 2: LINUX, WINDOWS EMBEDDED, ...



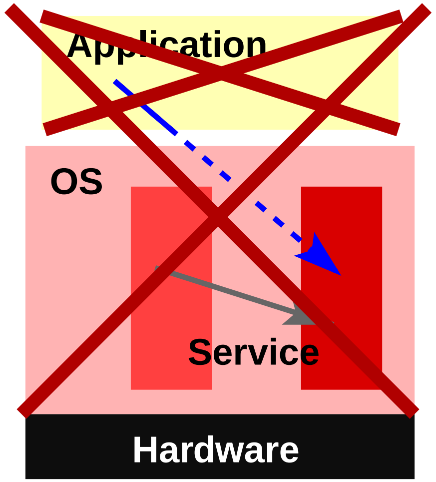
- Scaled-down version of desktop operating system
  - ➜ operating system protected from application misbehaviour

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 $\left(\stackrel{\mathrm{def}}{=} \text{part of system that executes in privileged mode}\right)$

➜ 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 $\left(\stackrel{\mathrm{def}}{=}\text{ part of system that executes in privileged mode}\right)$
  ➜ everything running in privileged mode can bypass security
- Device drivers
  ➜ DMA-capable devices can bypass protection
  ➜ drivers can mount DoS attacks
- Services that control resources
  ➜ resource owner can deny resource
  ➜ resource owner can leak/corrupt data

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
- Services that control resources
  - ➜ resource owner can deny resource
  - ➜ resource owner can leak/corrupt data
- *Everything* on MPU-less processors
  - ✗ no memory-protection hardware $\Rightarrow$ 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

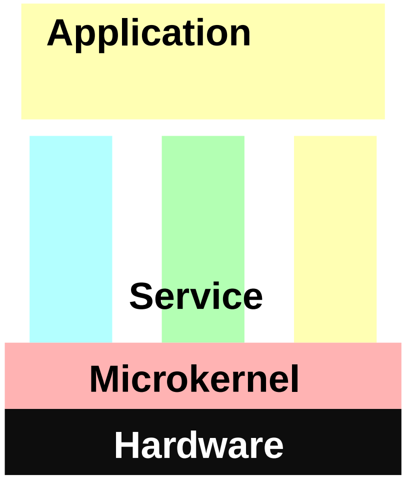
... MEANS FIRST OF ALL:

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

MINIMISING KERNEL SIZE:

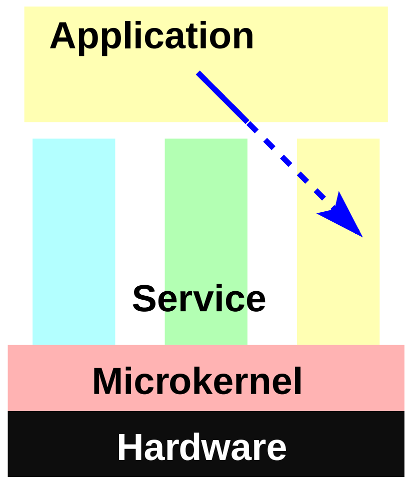
- Reduce kernel to what is *essential for supporting secure systems*
- What does not *require* privileged mode *must not* be in the kernel
- This is the definition of a *microkernel*
- *Minimal TCB required* $\Rightarrow$ *microkernel required!*

NICTA APPROACH: MICROKERNEL



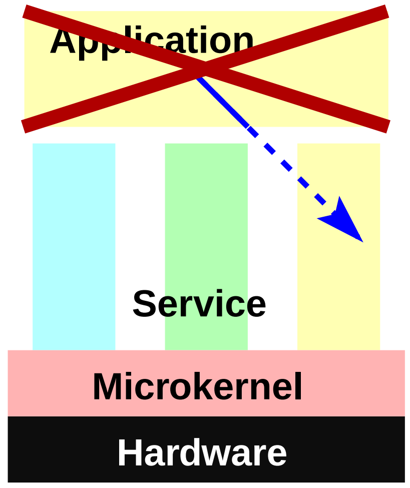
- Extremely small kernel
  - ➜ microkernel only contains code that *must* run in privileged mode
  - ➜ all other “systems” code runs as unprivileged servers

NICTA APPROACH: MICROKERNEL



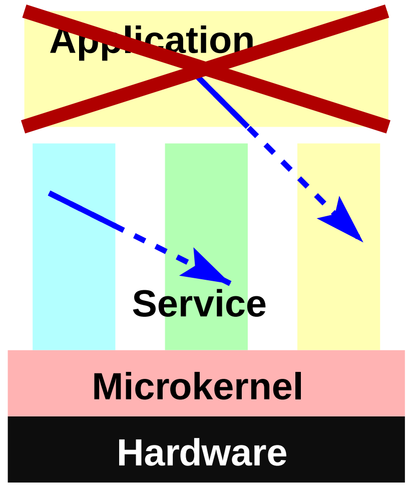
- Extremely small kernel
  - ➜ microkernel only contains code that *must* run in privileged mode
  - ➜ all other “systems” code runs as unprivileged servers

NICTA APPROACH: MICROKERNEL



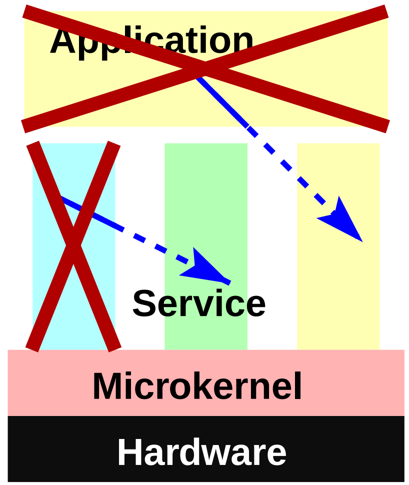
- Extremely small kernel
  - ➜ microkernel only contains code that *must* run in privileged mode
  - ➜ all other “systems” code runs as unprivileged servers

NICTA APPROACH: MICROKERNEL



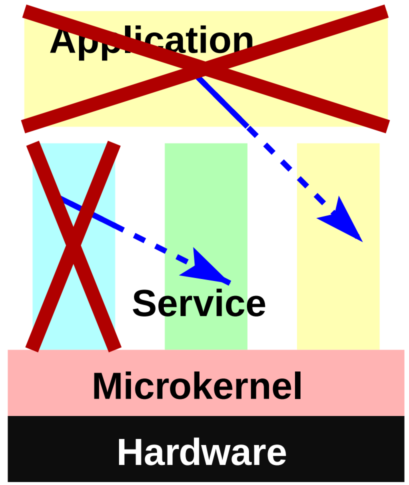
- Extremely small kernel
  - ➜ microkernel only contains code that *must* run in privileged mode
  - ➜ all other “systems” code runs as unprivileged servers

NICTA APPROACH: MICROKERNEL



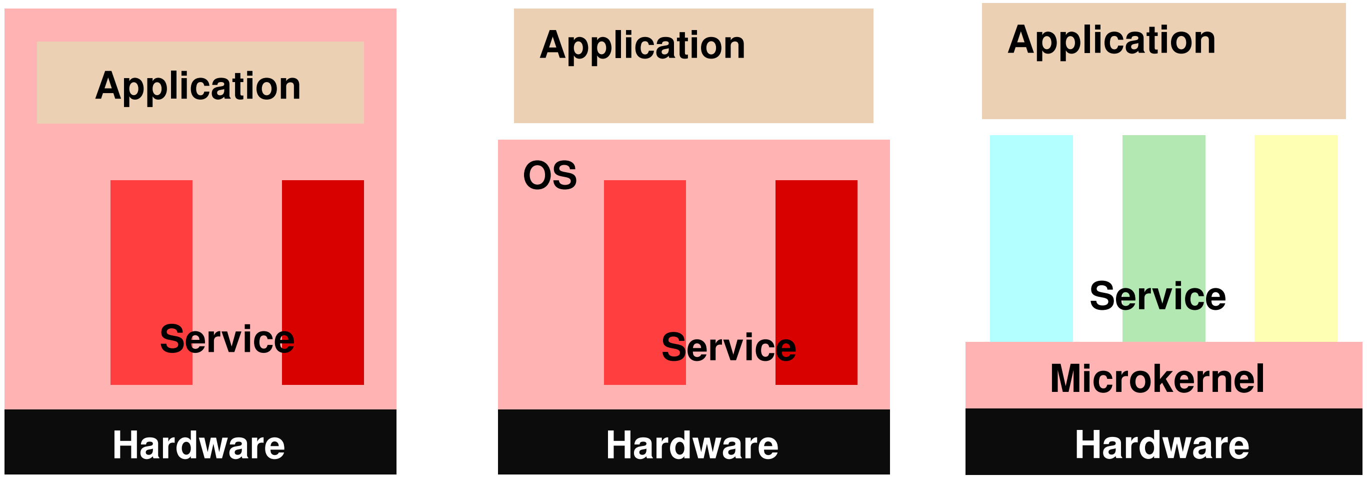
- Extremely small kernel
  - ➜ microkernel only contains code that *must* run in privileged mode
  - ➜ all other "systems" code runs as unprivileged servers
  - ➜ microkernel protected from application *and* other systems code
  - ➜ microkernel provides protection of all components from each other

NICTA APPROACH: MICROKERNEL



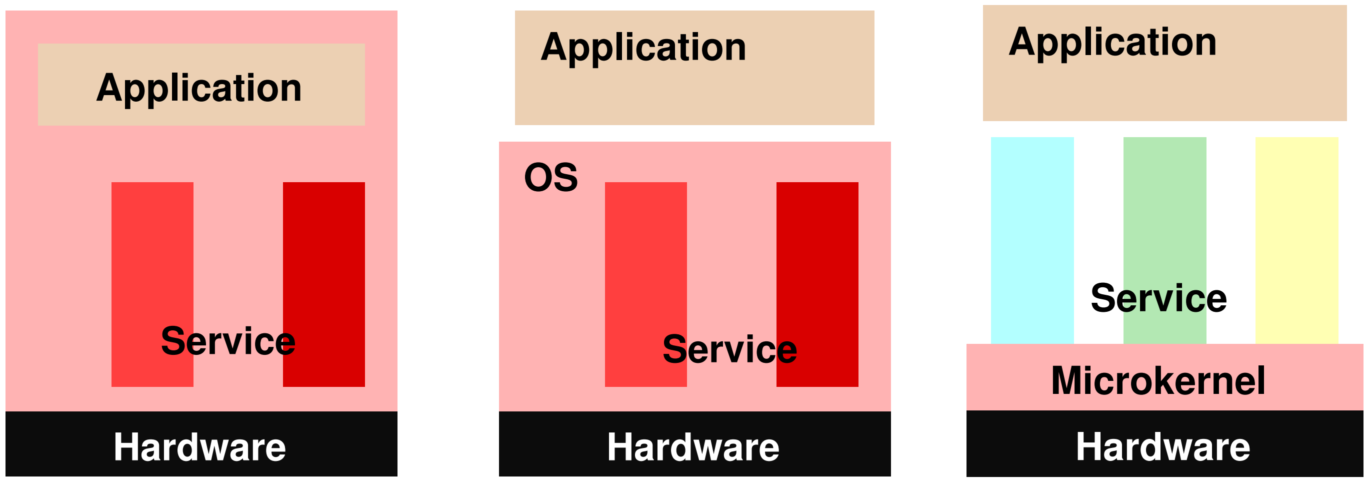
- Extremely small kernel
  - ➜ microkernel only contains code that *must* run in privileged mode
  - ➜ all other "systems" code runs as unprivileged servers
  - ➜ microkernel protected from application *and* other systems code
  - ➜ microkernel provides protection of all components from each other
- What's the difference?

TRUSTED COMPUTING BASE



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

Trusted Computing Base

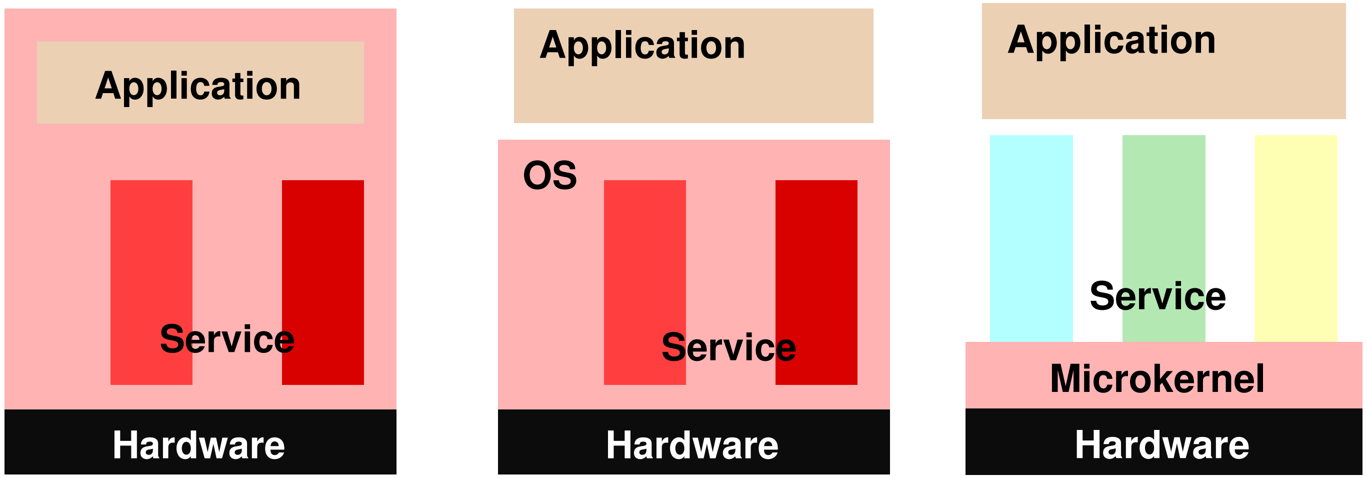


System:	traditional embedded	Linux/ Windows	Microkernel-based
TCB:	**all** code	100,000's loc	10,000's loc

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

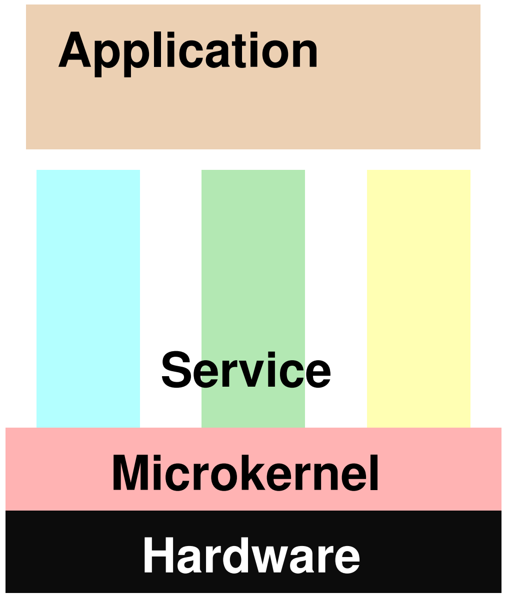
Small is beautiful:

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

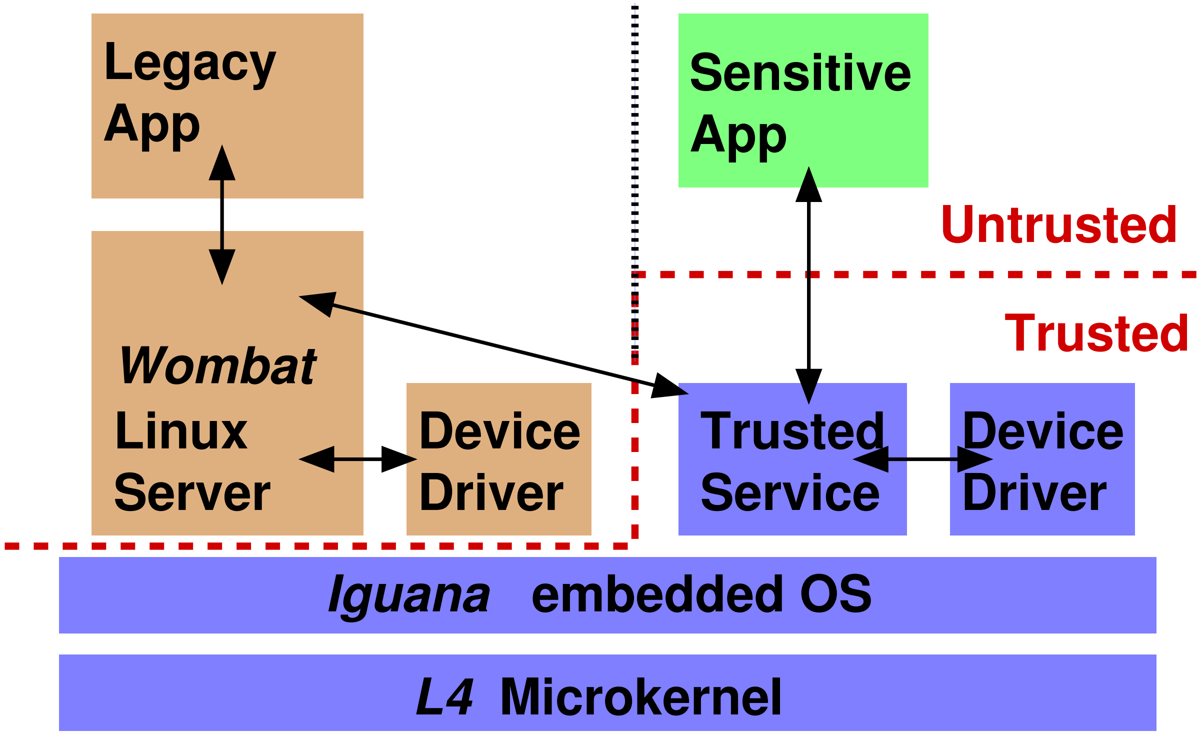
Challenge: Can we *guarantee* the trustworthiness of the TCB?

MICROKERNEL BENEFITS

- Fault isolation
  - ☑ other components protected from fault
- Hot swapping / hot upgrade
  - ☑ can replace servers on running system
- Software engineering techniques
  - ☑ componentised system
- Enforcement of security policies
  - ☑ kernel controls communication even between system services
- Resource management for system services
  - ☑ OS servers are like normal user code
- Formal verification
  - ☑ hardware-enforced isolation reduces complexity

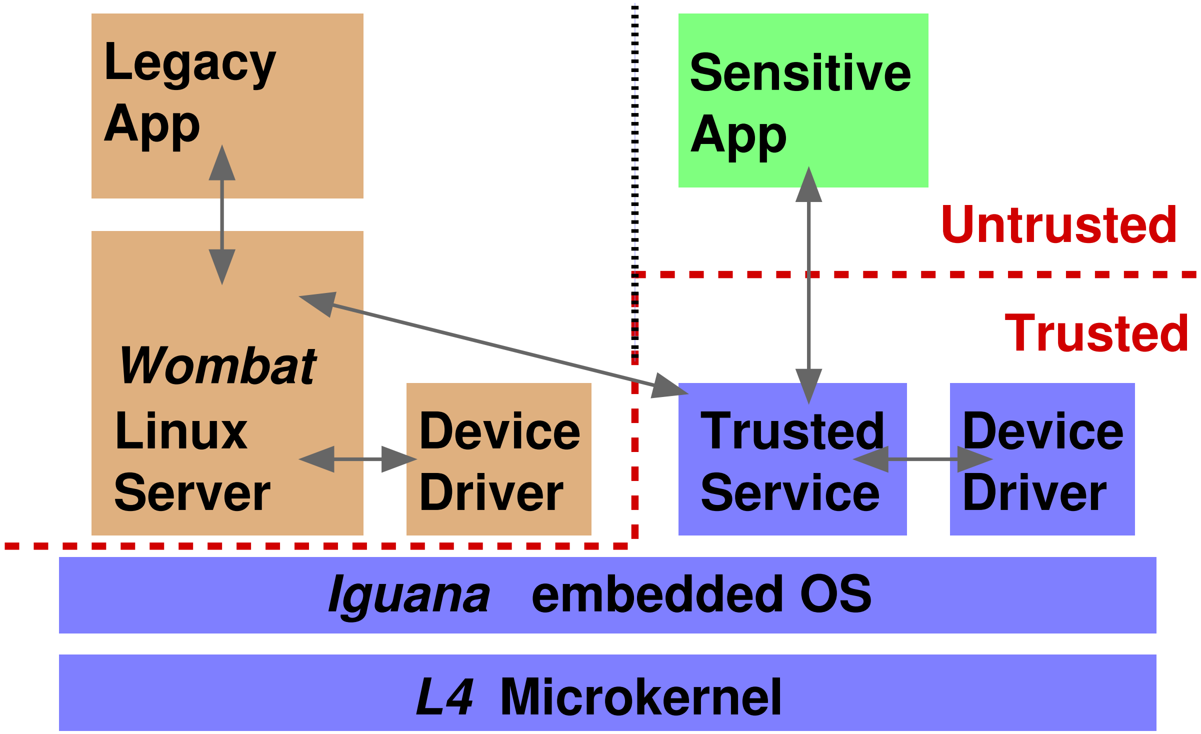


A SAMPLE SYSTEM



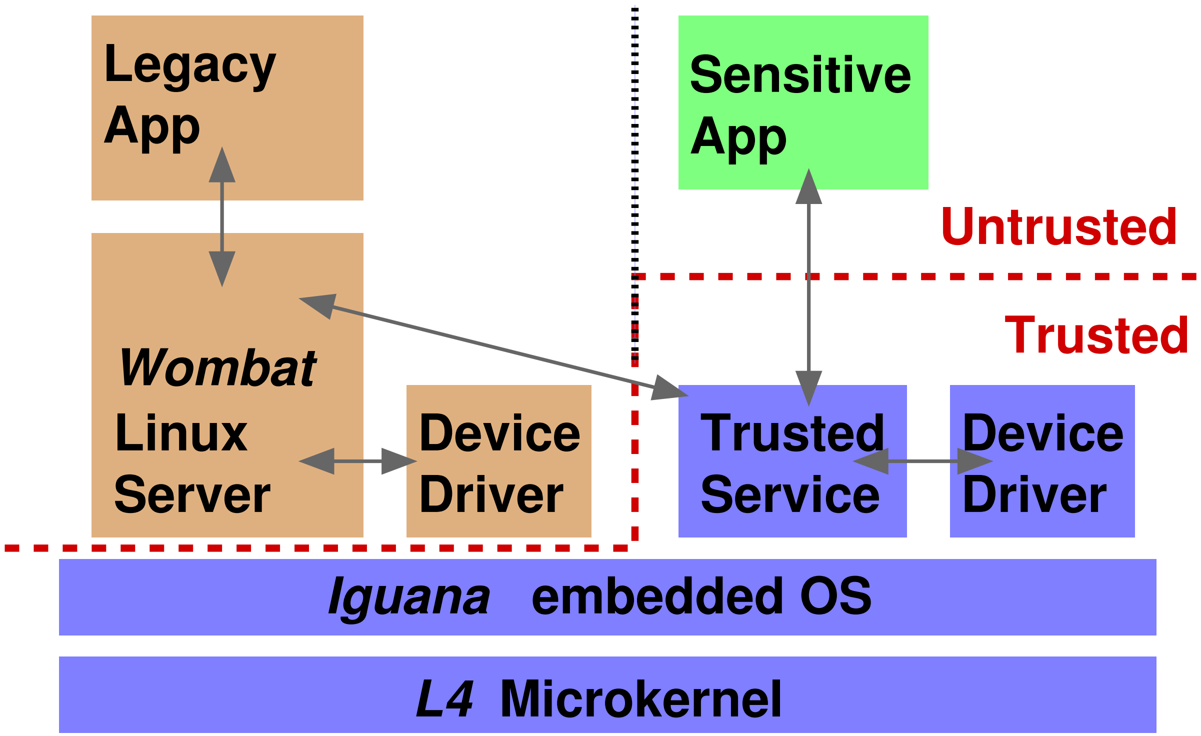
- Sensitive part of system has small TCB

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

L4: NICTA L4-EMBEDDED API

SIMPLER/SMALLER IMPLEMENTATION

- L4Ka::Pistachio issues
  - ⋆ optimised for supporting high-end systems (servers)
  - ⋆ large memory footprint (presently $\approx$ 100kB)
  - ➜ should fit on smartcard

L4: NICTA L4-EMBEDDED API

SIMPLER/SMALLER IMPLEMENTATION

- L4Ka::Pistachio issues
  - ⋆ optimised for supporting high-end systems (servers)
  - ⋆ large memory footprint (presently $\approx$ 100kB)
    - ➜ should fit on smartcard
  - ⋆ complexity (in-kernel page faults, preemption, code bloat)
    - ➜ makes it hard to validate implementation (formally or informally)
    - ➜ makes it hard to establish hard worst-case latencies
    - ➜ harder to optimise for high performance

L4: NICTA L4-EMBEDDED API

SIMPLER/SMALLER IMPLEMENTATION

- L4Ka::Pistachio issues
  - ⋆ optimised for supporting high-end systems (servers)
  - ⋆ large memory footprint (presently ≈ 100kB)
    - ➜ should fit on smartcard
  - ⋆ complexity (in-kernel page faults, preemption, code bloat)
    - ➜ makes it hard to validate implementation (formally or informally)
    - ➜ makes it hard to establish hard worst-case latencies
    - ➜ harder to optimise for high performance

- NICTA::Pistachio, implements L4 N1 API

L4: NICTA L4-EMBEDDED API

SIMPLER/SMALLER IMPLEMENTATION

- L4Ka::Pistachio issues
  - ⋆ optimised for supporting high-end systems (servers)
  - ⋆ large memory footprint (presently ≈ 100kB)
    - ➜ should fit on smartcard
  - ⋆ complexity (in-kernel page faults, preemption, code bloat)
    - ➜ makes it hard to validate implementation (formally or informally)
    - ➜ makes it hard to establish hard worst-case latencies
    - ➜ harder to optimise for high performance

- NICTA::Pistachio, implements L4 N1 API
  - ➜ remove long IPC!
  - ➜ reduce number of virtual registers (16–64)
  - ➜ remove timeouts
  - ➜ remove local thread IDs
  - ➜ remove remaining drivers from kernel

L4: NICTA L4-EMBEDDED API

SIMPLER/SMALLER IMPLEMENTATION

- L4Ka::Pistachio issues
  - ⋆ optimised for supporting high-end systems (servers)
  - ⋆ large memory footprint (presently ≈ 100kB)
    - ➜ should fit on smartcard
  - ⋆ complexity (in-kernel page faults, preemption, code bloat)
    - ➜ makes it hard to validate implementation (formally or informally)
    - ➜ makes it hard to establish hard worst-case latencies
    - ➜ harder to optimise for high performance

- NICTA::Pistachio, implements L4 N1 API
  - ➜ remove long IPC!
  - ➜ reduce number of virtual registers (16–64)
  - ➜ remove timeouts
  - ➜ remove local thread IDs
  - ➜ remove remaining drivers from kernel
  - ➜ add asynchronous notification

L4: NICTA L4-EMBEDDED API

SIMPLER/SMALLER IMPLEMENTATION

- L4Ka::Pistachio issues
  - ⋆ optimised for supporting high-end systems (servers)
  - ⋆ large memory footprint (presently $\approx$ 100kB)
    - ➜ should fit on smartcard
  - ⋆ complexity (in-kernel page faults, preemption, code bloat)
    - ➜ makes it hard to validate implementation (formally or informally)
    - ➜ makes it hard to establish hard worst-case latencies
    - ➜ harder to optimise for high performance

- NICTA::Pistachio, implements L4 N1 API
  - ➜ remove long IPC!
  - ➜ reduce number of virtual registers (16–64)
  - ➜ remove timeouts
  - ➜ remove local thread IDs
  - ➜ remove remaining drivers from kernel
  - ➜ add asynchronous notification
  - ➜ physically-addressed TCBs $\Leftarrow$ *honours thesis*
  - ➜ single kernel stack per CPU $\Leftarrow$ *honours project*

IGUANA: OS PERSONALITY FOR EMBEDDED SYSTEMS

- Remember, L4 is (almost) a “strict” microkernel:
  - ➜ does not provide any services
  - ➜ does not provide policies (or only very few)
  - ➜ provides mechanisms

- L4 aspires to be generic kernel, suitable for all kinds of systems

IGUANA: OS PERSONALITY FOR EMBEDDED SYSTEMS

- Remember, L4 is (almost) a “strict” microkernel:
  - ➜ does not provide any services
  - ➜ does not provide policies (or only very few)
  - ➜ provides mechanisms
- L4 aspires to be generic kernel, suitable for all kinds of systems
- Almost any system requires a set of core services:
  - ➜ process management
  - ➜ memory management
  - ➜ security management

  ... based on some system-wide policies

IGUANA: OS PERSONALITY FOR EMBEDDED SYSTEMS

- Remember, L4 is (almost) a “strict” microkernel:
  - ➜ does not provide any services
  - ➜ does not provide policies (or only very few)
  - ➜ provides mechanisms
- L4 aspires to be generic kernel, suitable for all kinds of systems
- Almost any system requires a set of core services:
  - ➜ process management
  - ➜ memory management
  - ➜ security management

  ... based on some system-wide policies
- Iguana provides these (or at least more tools for providing them)
  - ➜ designed for use in embedded systems
  - ➜ designed to minimise trusted computing base

What does Iguana Provide?

- Convenient way of using L4 primitives
  - ➜ OO-style method invocations instead of explicit IPC calls
  - ➜ IDL compiler for automatic generation of stubs

What does Iguana Provide?

- Convenient way of using L4 primitives
  - OO-style method invocations instead of explicit IPC calls
  - IDL compiler for automatic generation of stubs
- Protection framework for access rights management
  - capability based, flexible
  - able to model most standard security models

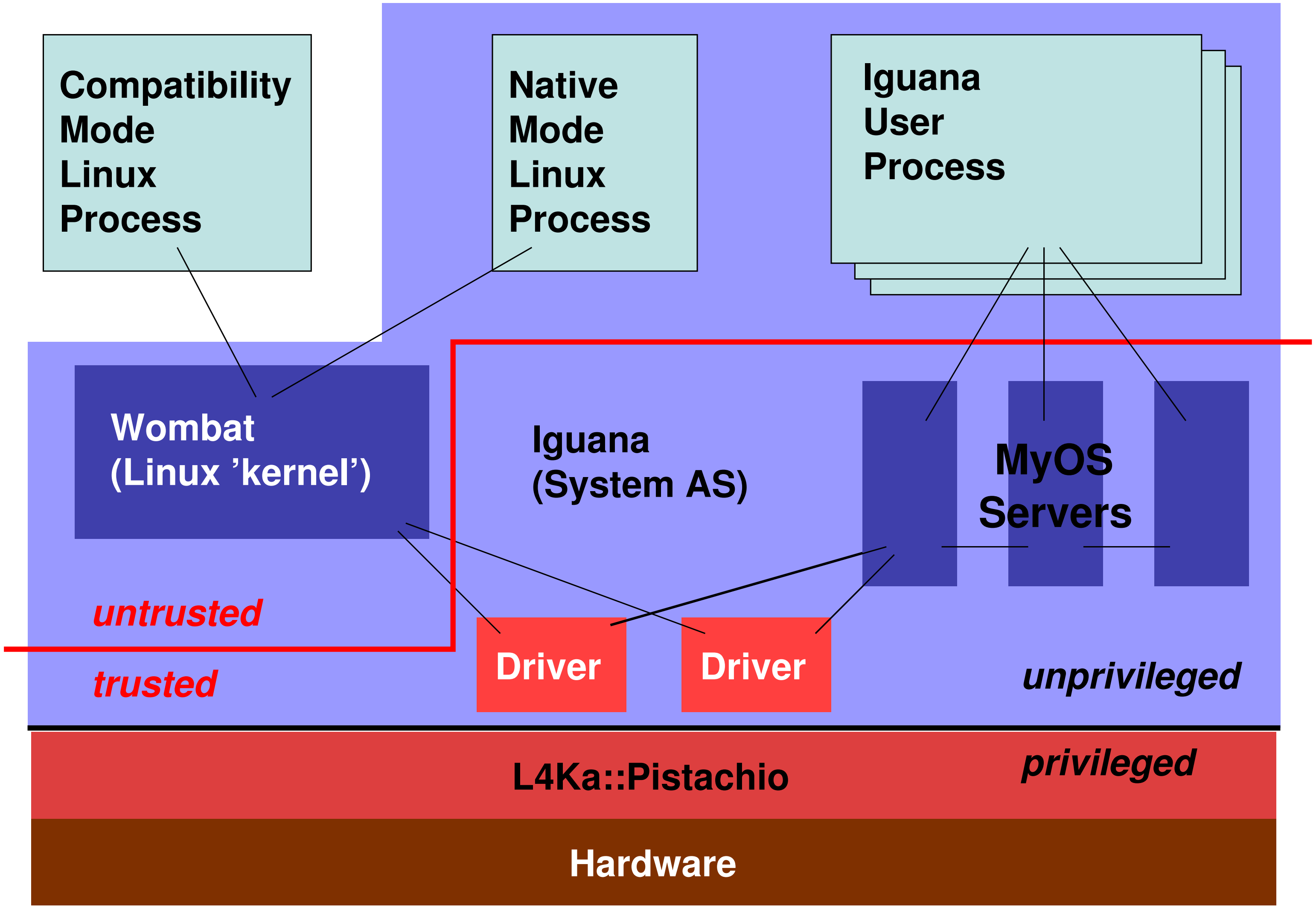
What does Iguana Provide?

- ⋆ Convenient way of using L4 primitives
  - ➜ OO-style method invocations instead of explicit IPC calls
  - ➜ IDL compiler for automatic generation of stubs
- ⋆ Protection framework for access rights management
  - ➜ capability based, flexible
  - ➜ able to model most standard security models
- ⋆ Virtual memory management
  - ➜ allocation, deallocation, sharing, ...
  - ➜ single-address-space view, supporting FASS on ARM

What does Iguana Provide?

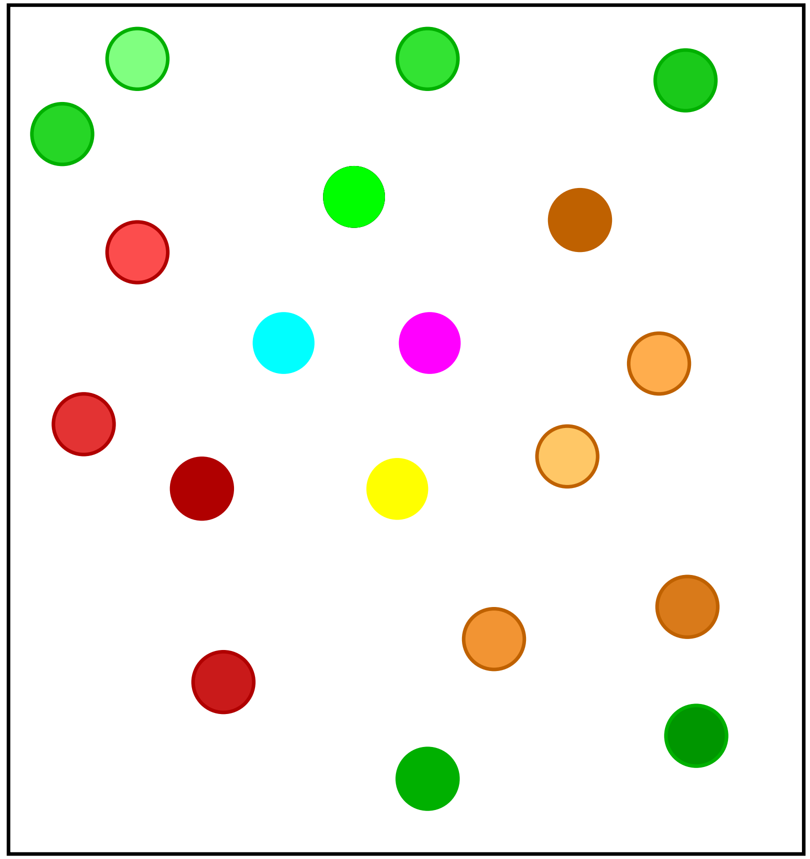
- ⋆ Convenient way of using L4 primitives
  - ➜ OO-style method invocations instead of explicit IPC calls
  - ➜ IDL compiler for automatic generation of stubs
- ⋆ Protection framework for access rights management
  - ➜ capability based, flexible
  - ➜ able to model most standard security models
- ⋆ Virtual memory management
  - ➜ allocation, deallocation, sharing, ...
  - ➜ single-address-space view, supporting FASS on ARM
- ⋆ Protection-domain (process) management
- ⋆ Thread management

SAMPLE IGUANA SYSTEM



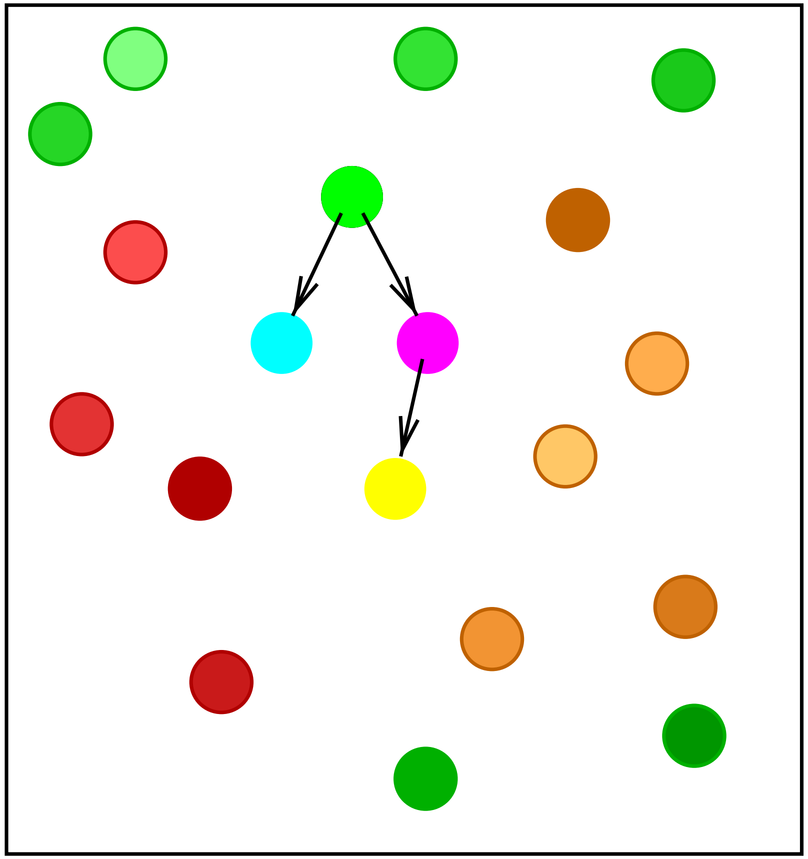
Iguana: Basic Approach

- Basic idea: single address space (SAS)



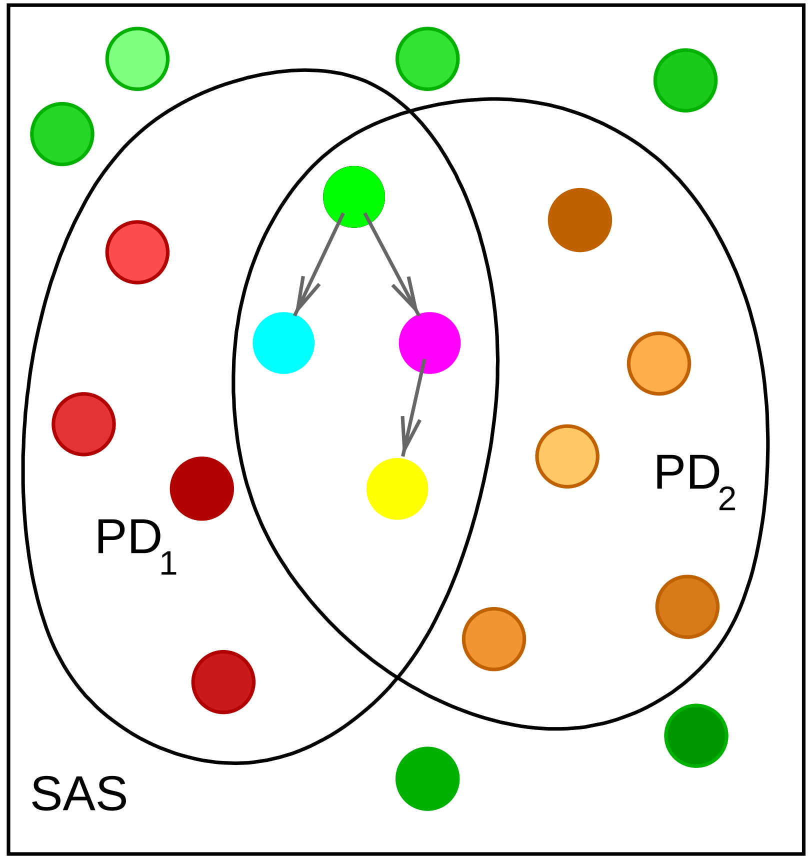
IGUANA: BASIC APPROACH

- Basic idea: single address space (SAS)
  - ⋆ eases sharing of data
  - ➜ minimises copying
  - ➜ no problems with pointers



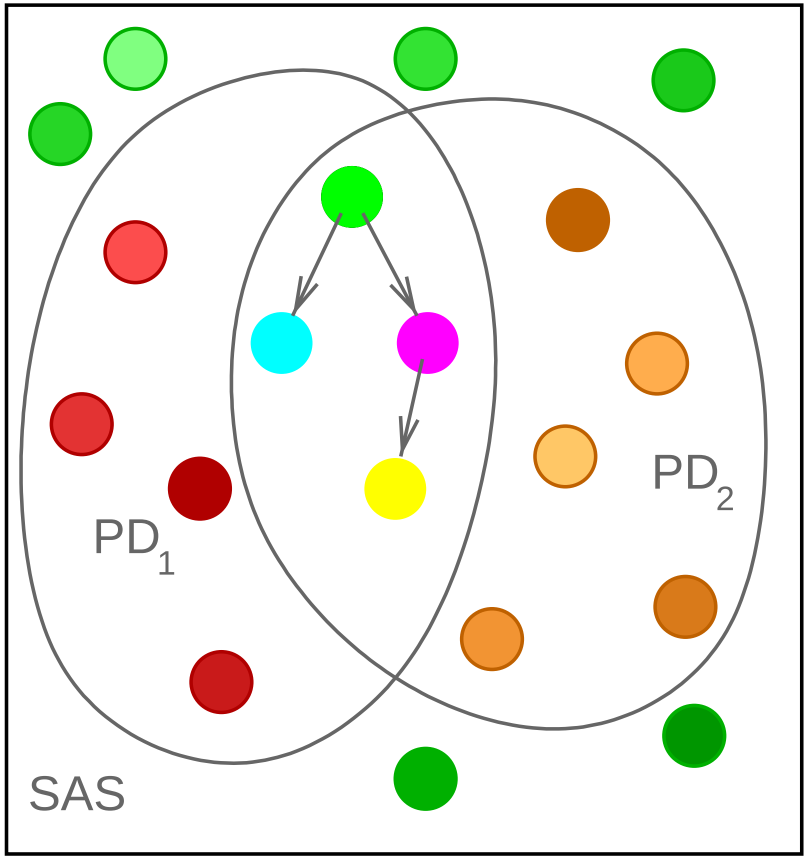
IGUANA: BASIC APPROACH

- Basic idea: single address space (SAS)
  - ⋆ eases sharing of data
    - ➜ minimises copying
    - ➜ no problems with pointers
- Per-process *protection domains*
  - ⋆ enforce security policy
    - ➜ any access is subject to access control
  - ⋆ do not interfere with sharing



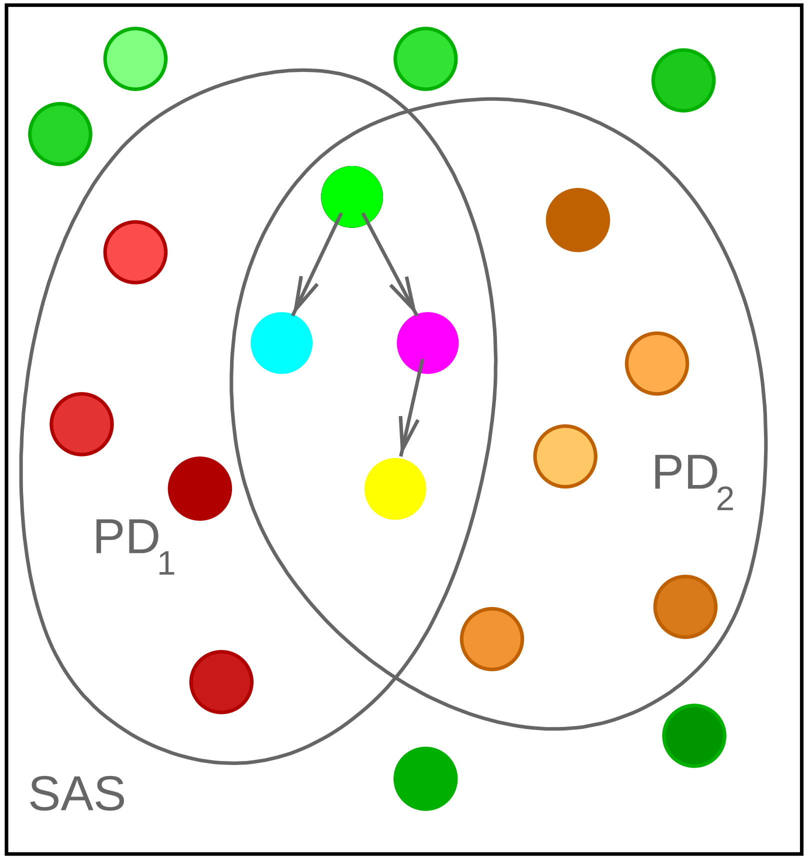
Iguana: Basic Approach

- Basic idea: single address space (SAS)
  - ⋆ eases sharing of data
    - ➜ minimises copying
    - ➜ no problems with pointers
- Per-process *protection domains*
  - ⋆ enforce security policy
    - ➜ any access is subject to access control
  - ⋆ do not interfere with sharing
- Advantages
  - ⋆ works with MMU-less microprocessors
  - ⋆ cheap context switches on ARM7/ARM9



IGUANA: BASIC APPROACH

- Basic idea: single address space (SAS)
  - ⋆ eases sharing of data
    - ➜ minimises copying
    - ➜ no problems with pointers
- Per-process *protection domains*
  - ⋆ enforce security policy
    - ➜ any access is subject to access control
  - ⋆ do not interfere with sharing
- Advantages
  - ⋆ works with MMU-less microprocessors
  - ⋆ cheap context switches on ARM7/ARM9
- Based on UNSW Mungi system



Iguana Concepts

- Memory section
  - ➜ unit of VM allocation and protection
  - ➜ can be an encapsulated object with methods and data

IGUANA CONCEPTS

- Memory section
  - ➜ unit of VM allocation and protection
  - ➜ can be an encapsulated object with methods and data
- Thread
  - ➜ execution abstraction, as in L4

Iguana Concepts

- Memory section
  - ➜ unit of VM allocation and protection
  - ➜ can be an encapsulated object with methods and data
- Thread
  - ➜ execution abstraction, as in L4
- Server
  - ➜ thread associated with memory section
  - ➜ invoked through methods with well-defined interfaces

IGUANA CONCEPTS

- Memory section
  - ➜ unit of VM allocation and protection
  - ➜ can be an encapsulated object with methods and data
- Thread
  - ➜ execution abstraction, as in L4
- Server
  - ➜ thread associated with memory section
  - ➜ invoked through methods with well-defined interfaces
- Protection domain
  - ➜ defines access and resource rights of a thread
  - ➜ corresponds to a process in traditional OS

IGUANA CONCEPTS

- Session
  - ➜ client-server (or peer-to-peer) communication channel
  - ➜ amortises authentication cost over many invocations

IGUANA CONCEPTS

- Session
  - ➜ client-server (or peer-to-peer) communication channel
  - ➜ amortises authentication cost over many invocations
- Capability
  - ➜ represents access rights
  - ➜ basis of protection

Iguana Concepts

- Session
  - ➜ client-server (or peer-to-peer) communication channel
  - ➜ amortises authentication cost over many invocations
- Capability
  - ➜ represents access rights
  - ➜ basis of protection
- Resource token
  - ➜ represents resource usage right
  - ➜ basis of resource management

IGUANA CONCEPTS

- Session
  - ➜ client-server (or peer-to-peer) communication channel
  - ➜ amortises authentication cost over many invocations
- Capability
  - ➜ represents access rights
  - ➜ basis of protection
- Resource token
  - ➜ represents resource usage right
  - ➜ basis of resource management
- External Space
  - ➜ address space extern to Iguana's SAS
  - ➜ for legacy support and large processes

Iguana Philosophy

- Small and lightweight
- Strong yet unintrusive protection
- Support for resource management
- Legacy support
- Code and concept re-use

Iguana Philosophy

- Small and lightweight
  - ➜ geared towards embedded systems
  - ➜ allow optimal utilisation of hardware
- Strong yet unintrusive protection
- Support for resource management
- Legacy support
- Code and concept re-use

IGUANA PHILOSOPHY

- Small and lightweight
  - ➜ geared towards embedded systems
  - ➜ allow optimal utilisation of hardware
- Strong yet unintrusive protection
  - ➜ hide protection machinery from most apps
  - ➜ able to emulate most standard protection models
- Support for resource management
- Legacy support
- Code and concept re-use

IGUANA PHILOSOPHY

- Small and lightweight
  - ➜ geared towards embedded systems
  - ➜ allow optimal utilisation of hardware
- Strong yet unintrusive protection
  - ➜ hide protection machinery from most apps
  - ➜ able to emulate most standard protection models
- Support for resource management
  - ➜ in principle, although it isn't implemented yet!
- Legacy support
- Code and concept re-use

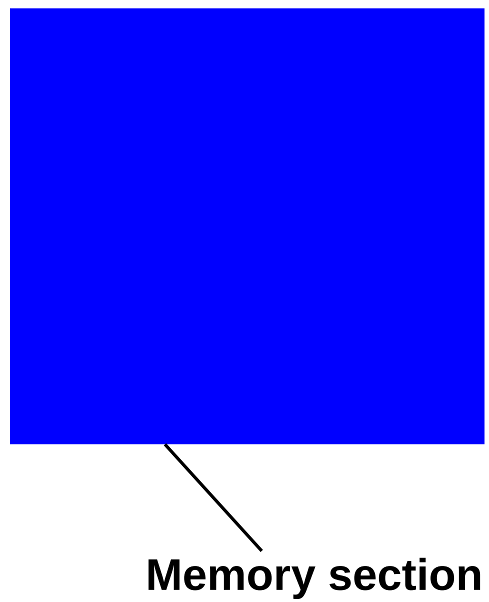
Iguana Philosophy

- Small and lightweight
  - ➜ geared towards embedded systems
  - ➜ allow optimal utilisation of hardware
- Strong yet unintrusive protection
  - ➜ hide protection machinery from most apps
  - ➜ able to emulate most standard protection models
- Support for resource management
  - ➜ in principle, although it isn't implemented yet!
- Legacy support
  - ➜ designed to run Linux server
- Code and concept re-use

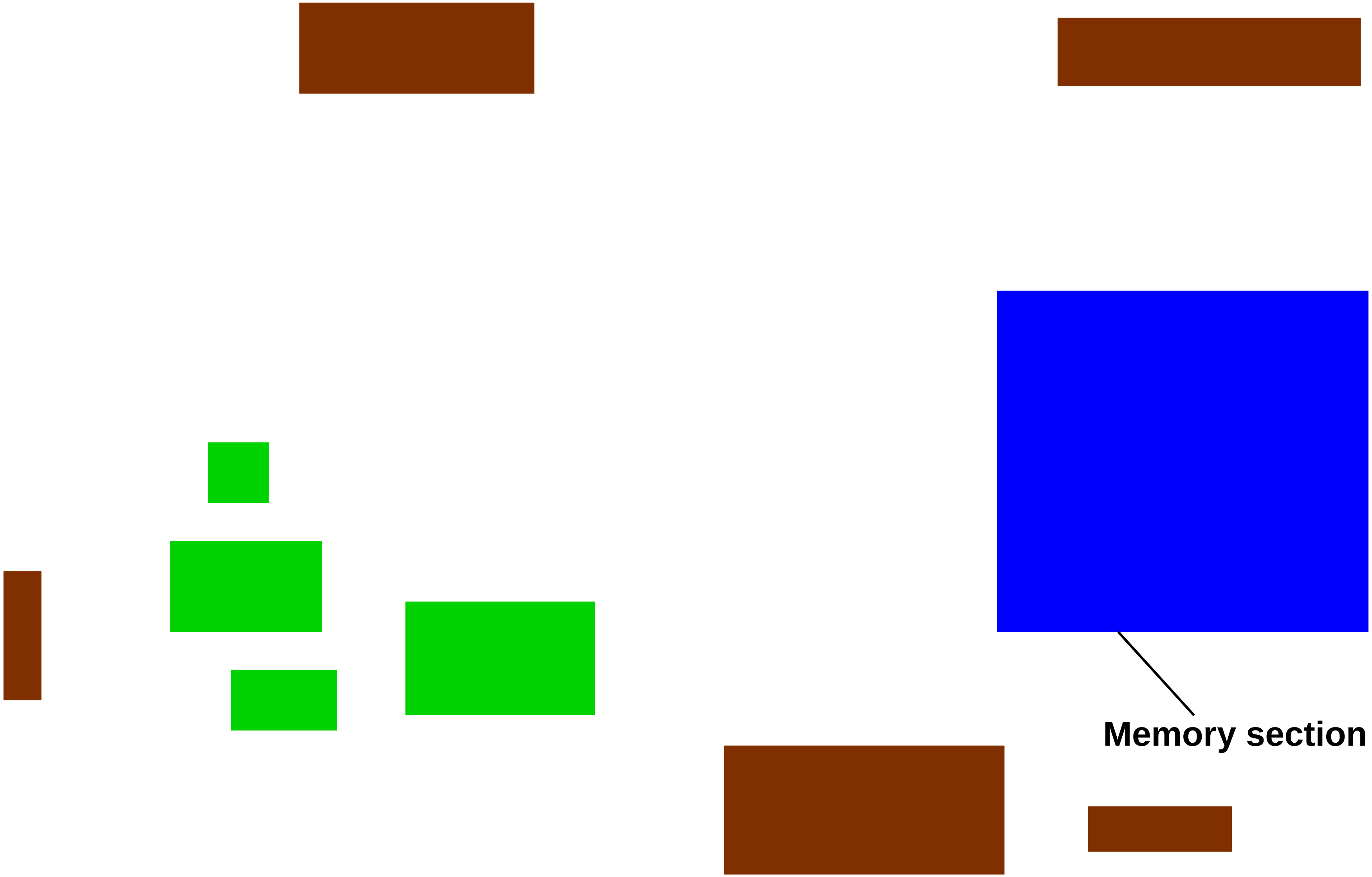
IGUANA PHILOSOPHY

- Small and lightweight
  - ➜ geared towards embedded systems
  - ➜ allow optimal utilisation of hardware
- Strong yet unintrusive protection
  - ➜ hide protection machinery from most apps
  - ➜ able to emulate most standard protection models
- Support for resource management
  - ➜ in principle, although it isn't implemented yet!
- Legacy support
  - ➜ designed to run Linux server
- Code and concept re-use
  - ➜ utilise what we've leaned in and developed for Mungi project

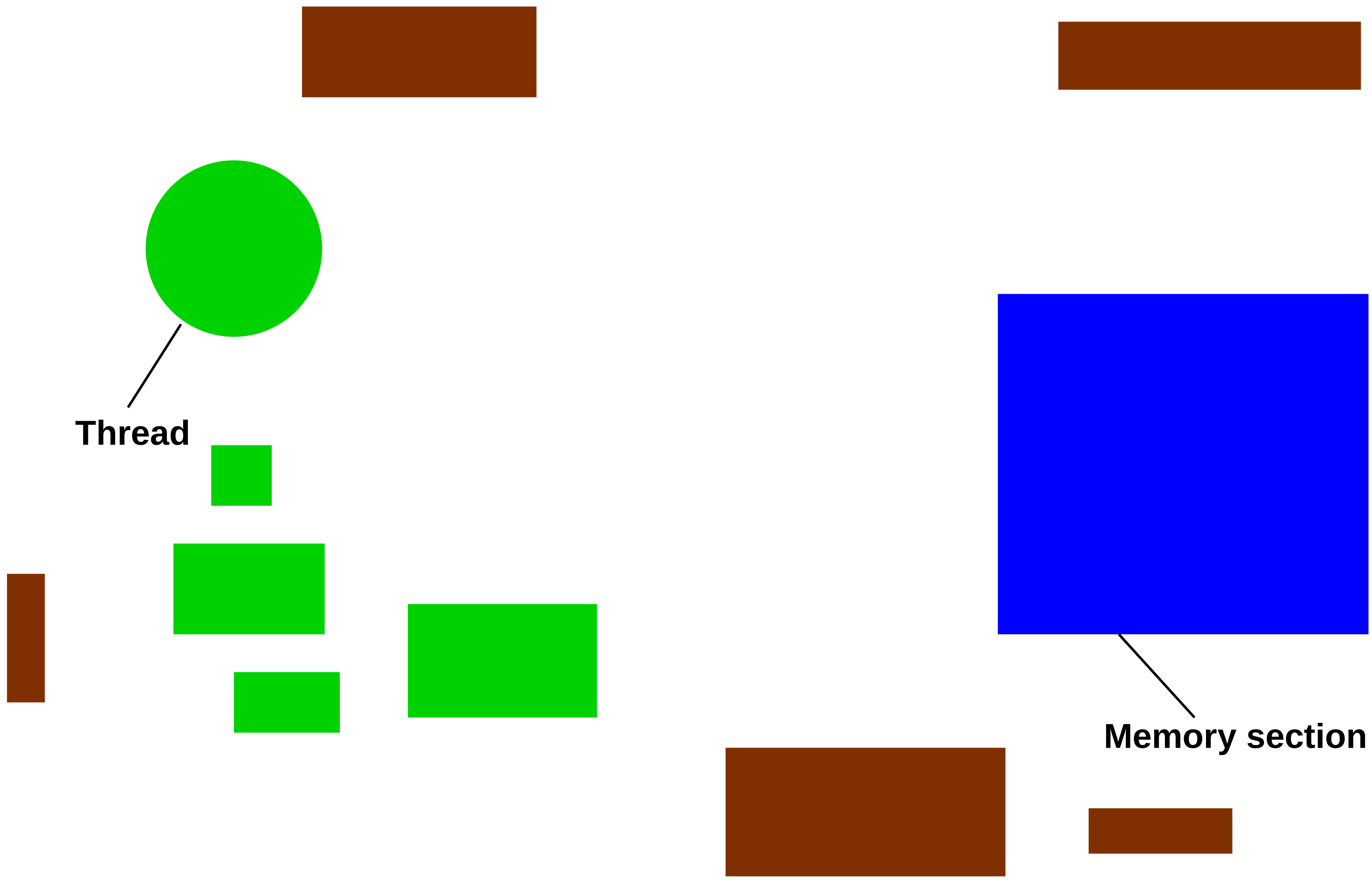
Iguana Concepts



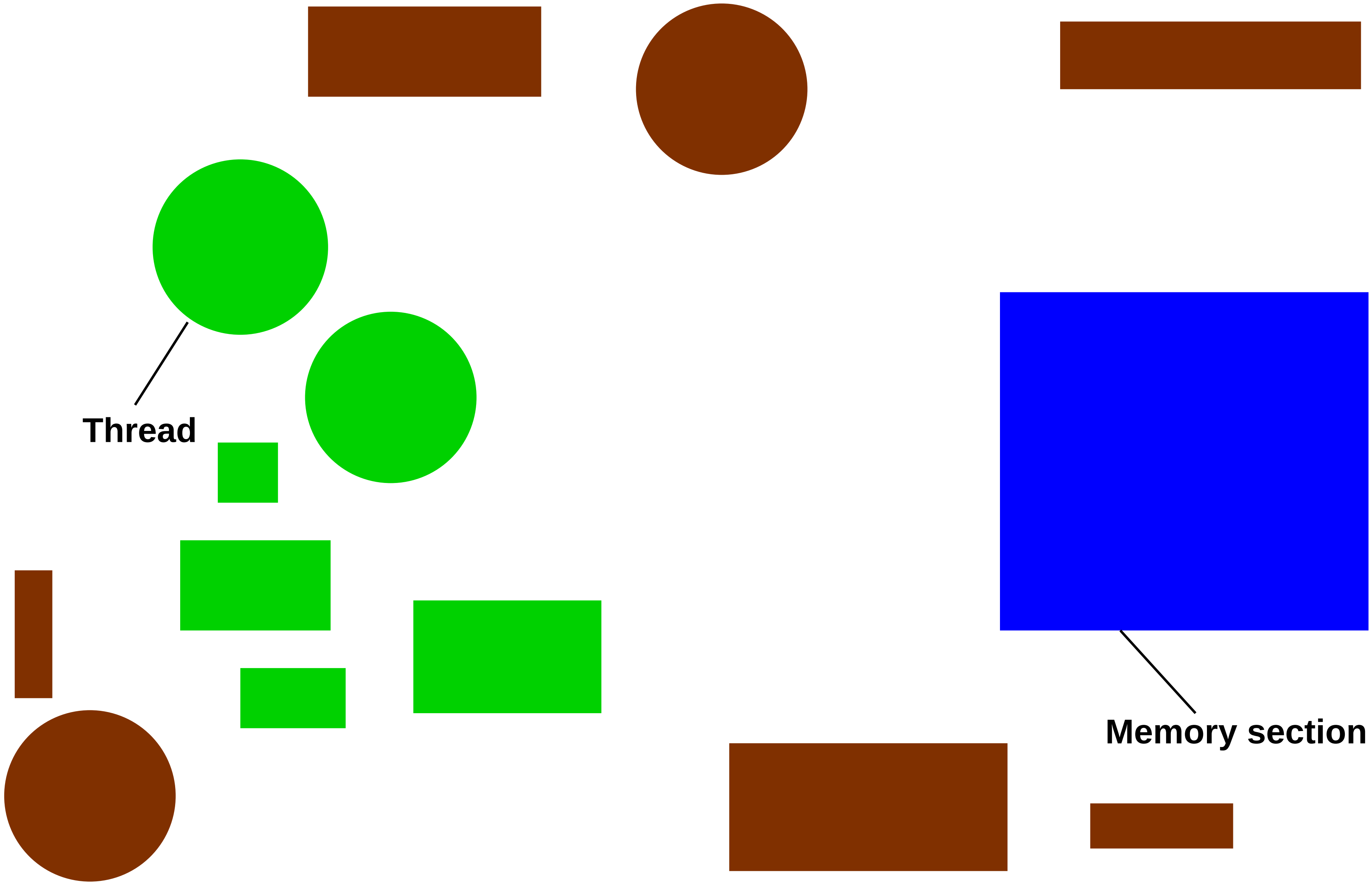
Iguana Concepts



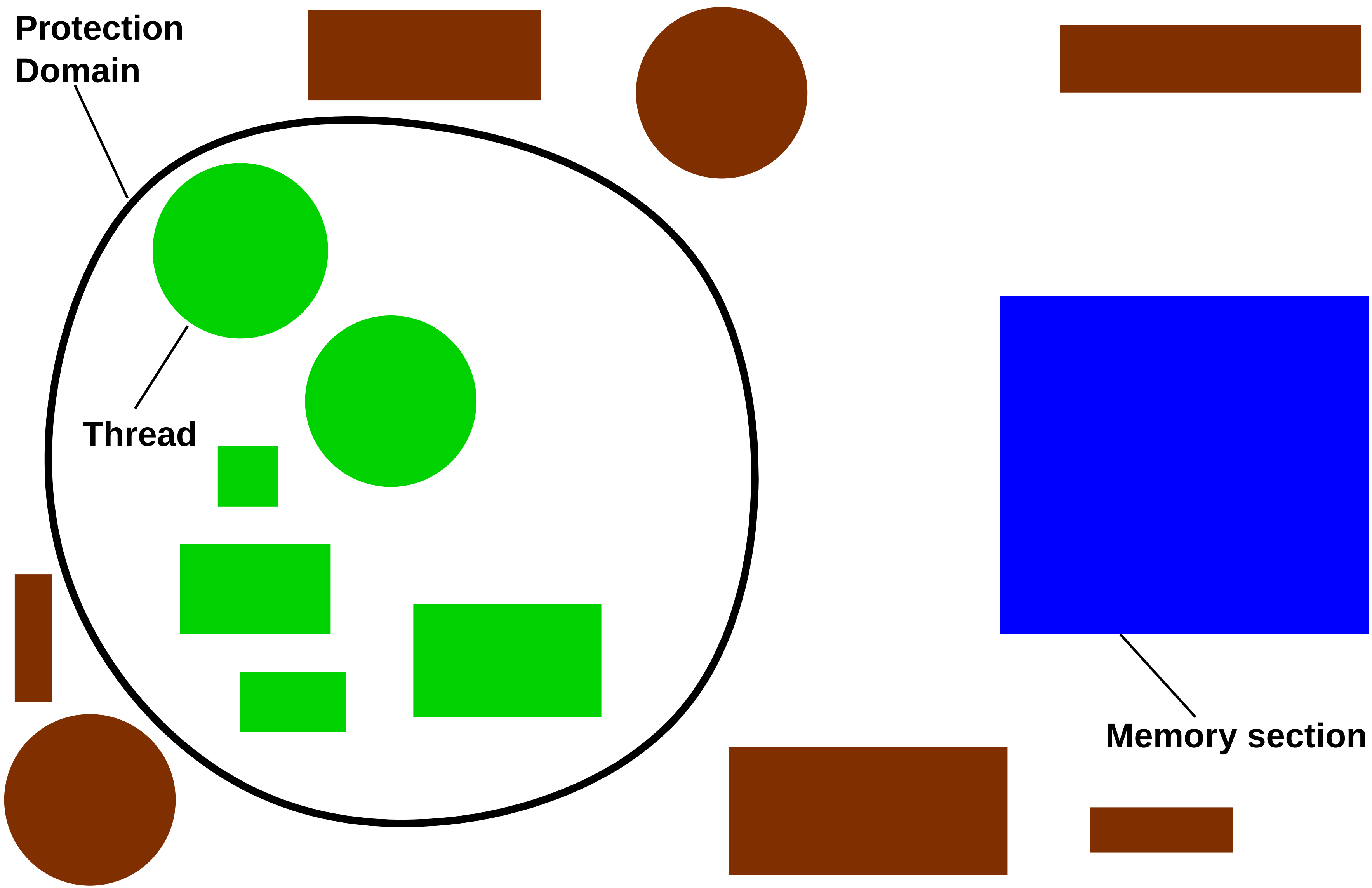
Iguana Concepts



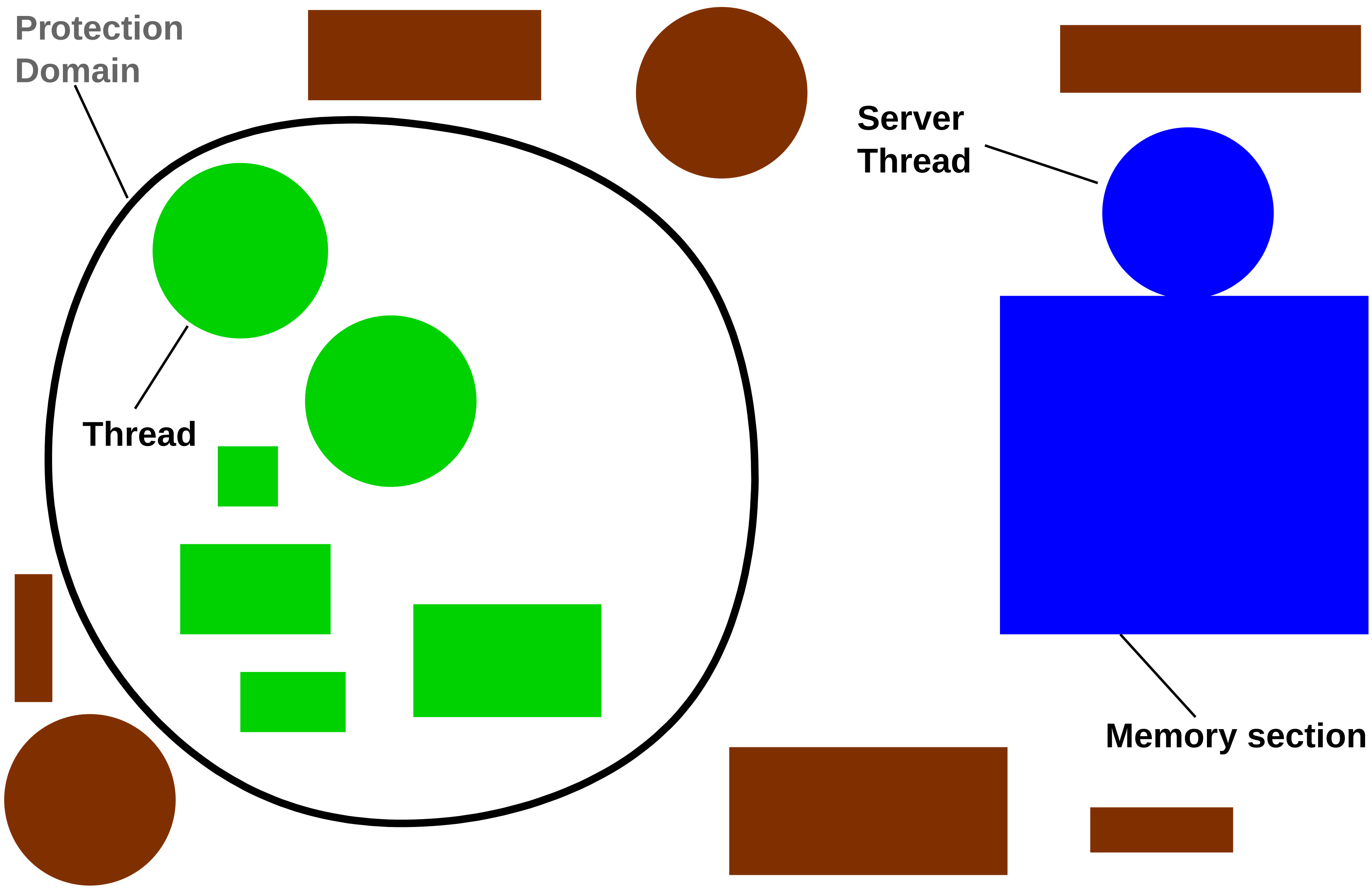
Iguana Concepts



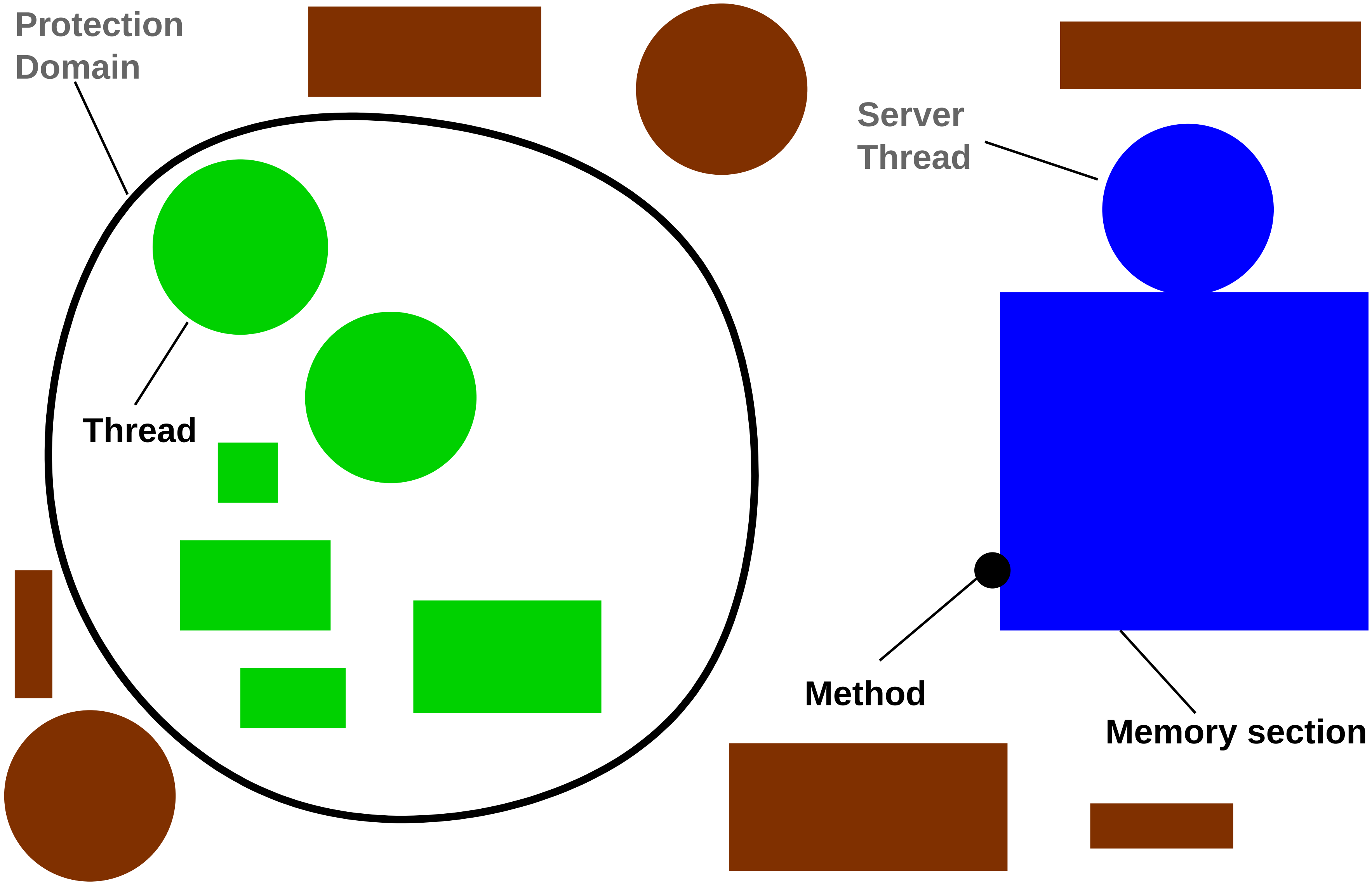
Iguana Concepts



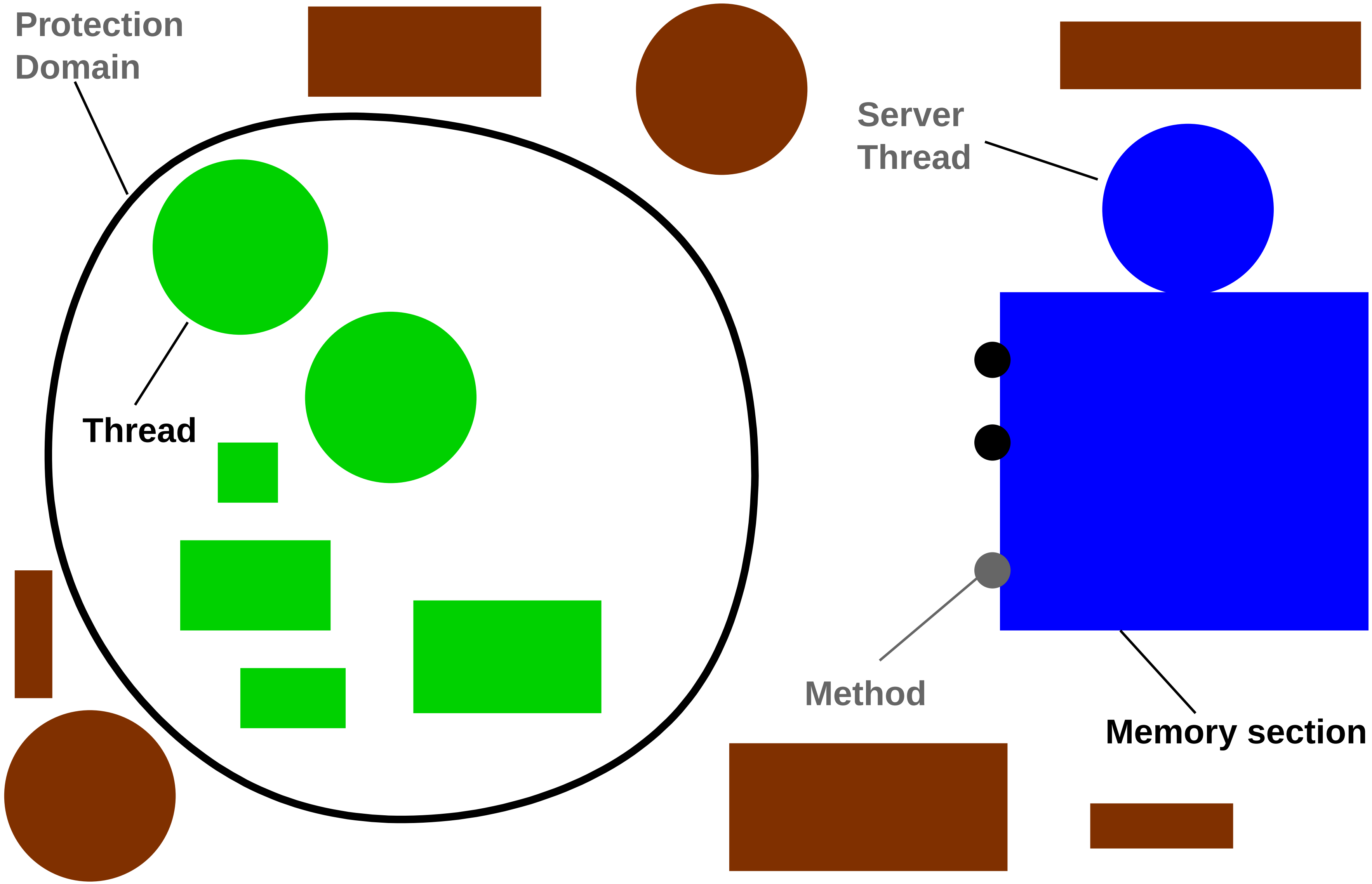
Iguana Concepts



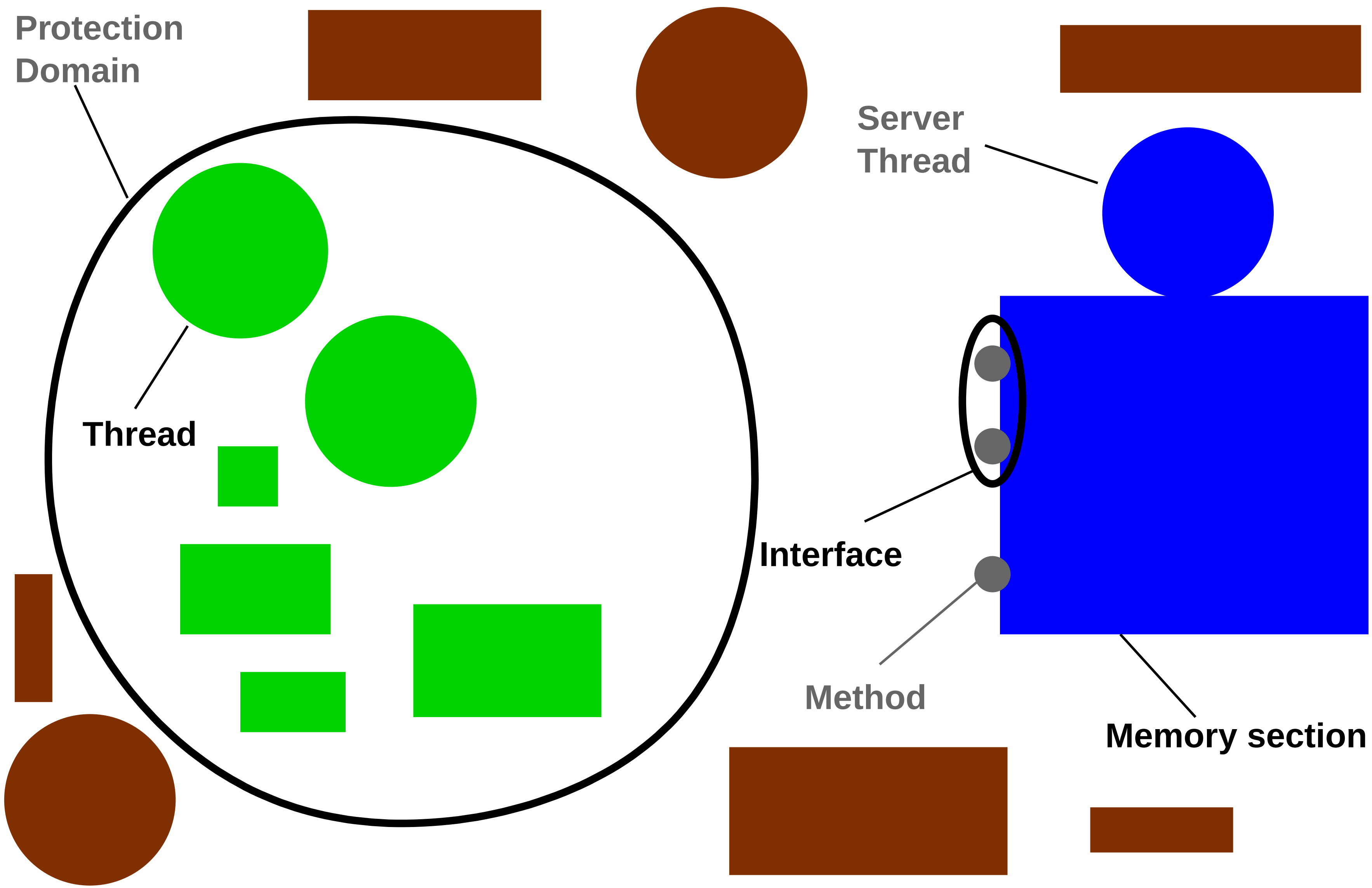
Iguana Concepts



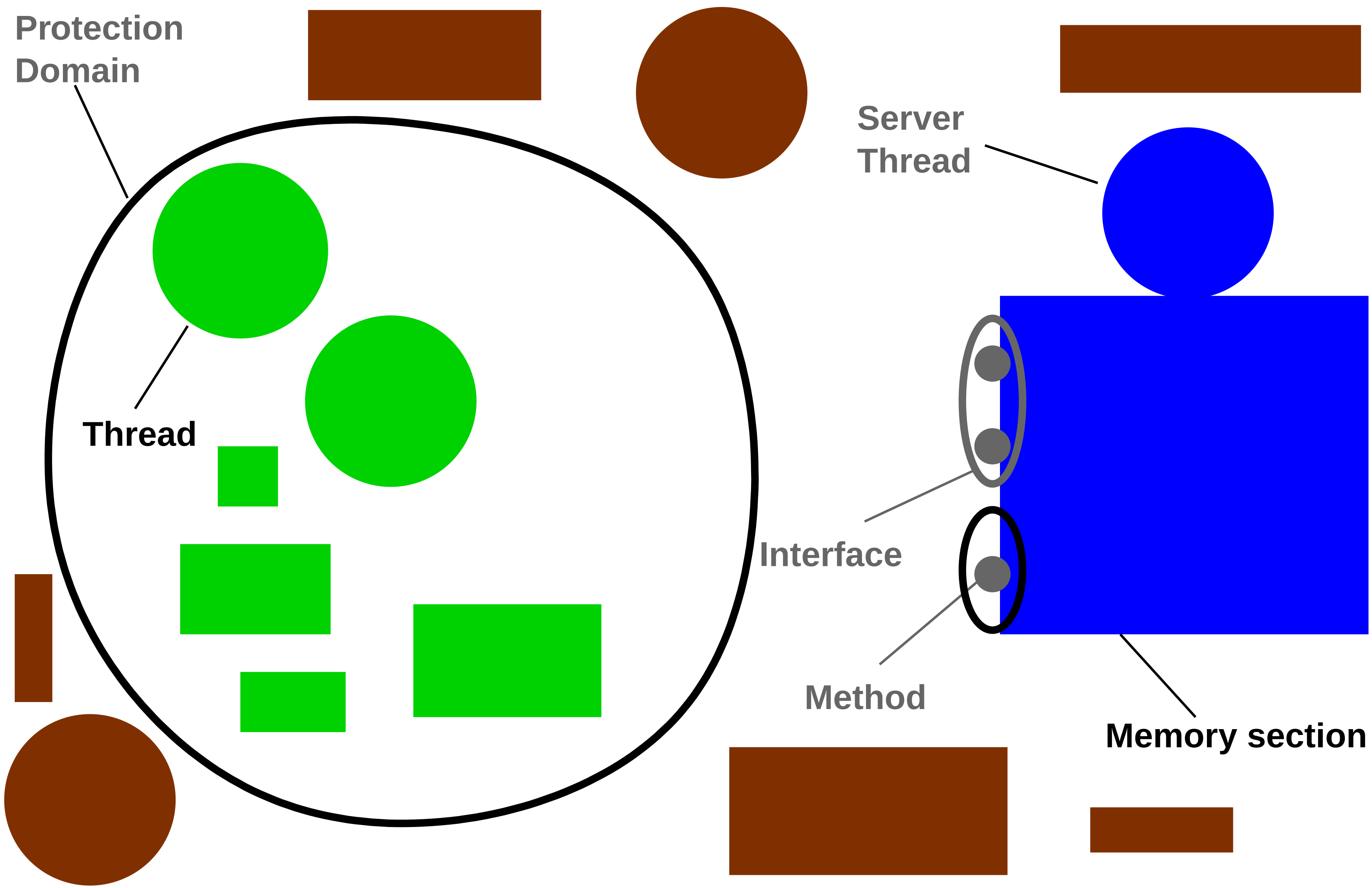
Iguana Concepts



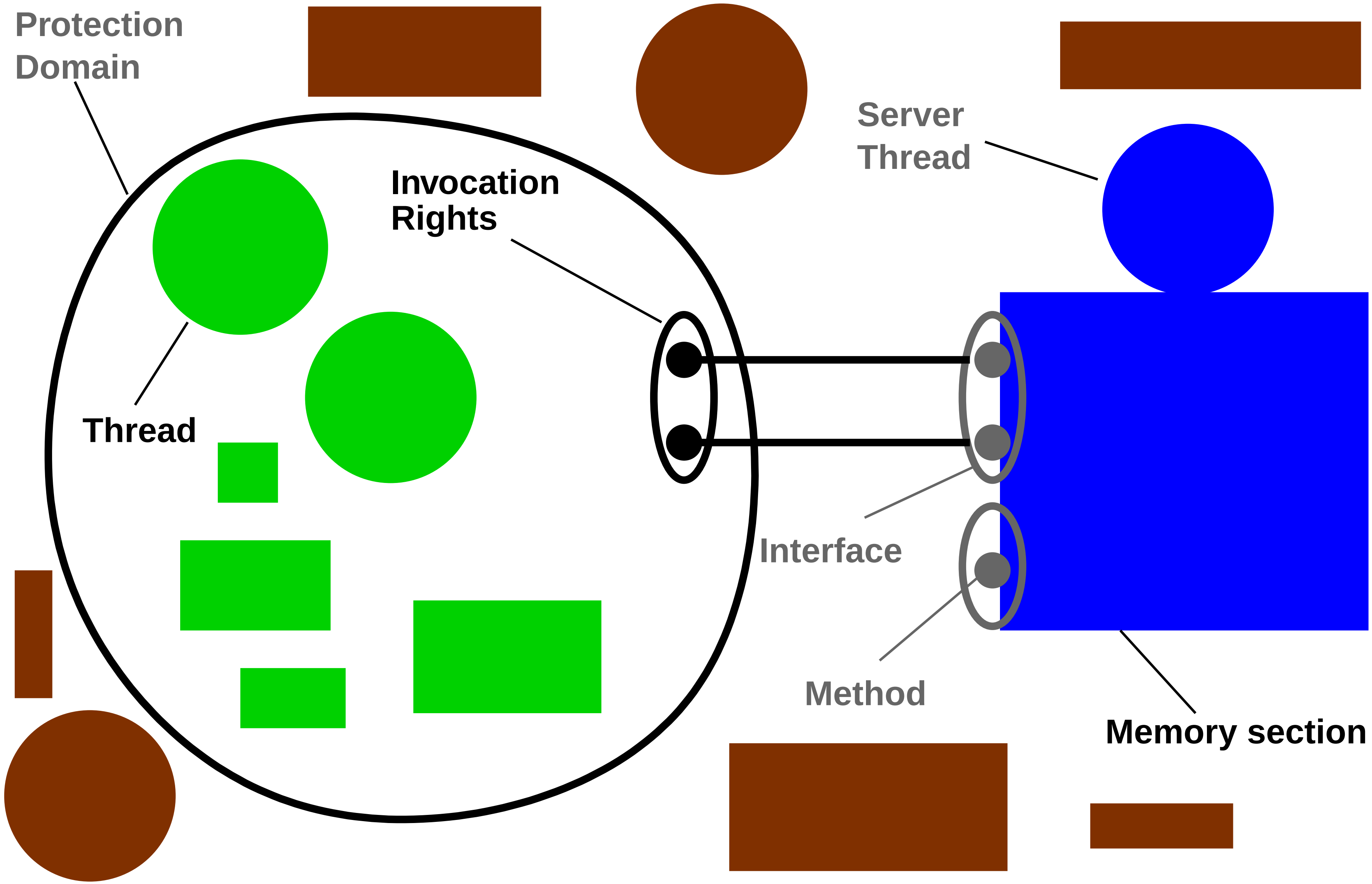
Iguana Concepts



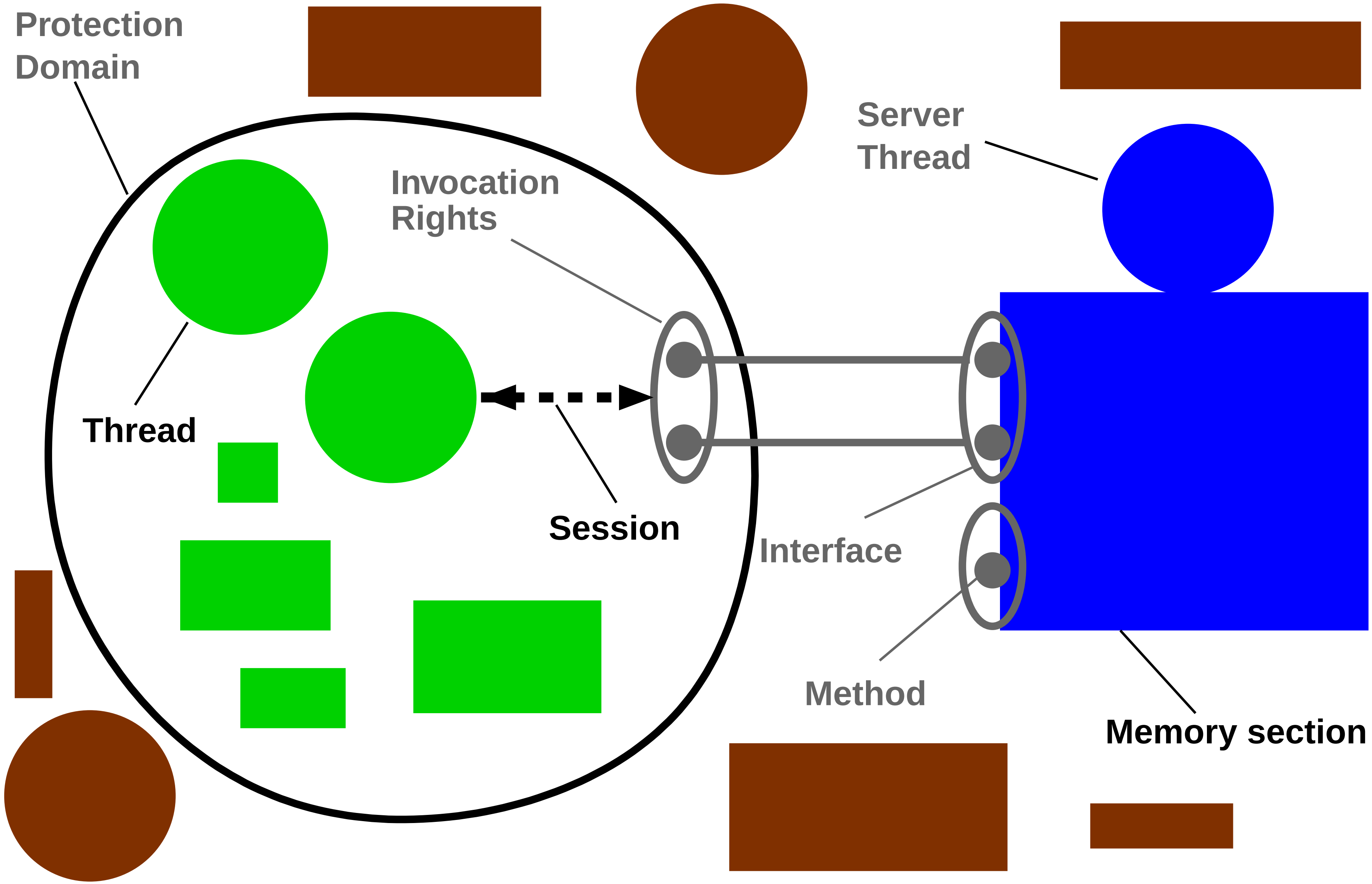
Iguana Concepts



Iguana Concepts



IGUANA CONCEPTS



Iguana Objects

- Six kinds of *objects*

1. memory sections
2. threads
3. protection domains (PDs)
4. sessions
5. resource tokens (restoks)
   - ➜ not yet implemented, not covered here
6. external spaces
   - ➜ not full Iguana objects
   - ➜ serve as proxies for non-Iguana objects

IGUANA OBJECTS

- Six kinds of *objects*
  1. memory sections
  2. threads
  3. protection domains (PDs)
  4. sessions
  5. resource tokens (restoks)
     - ➜ not yet implemented, not covered here
  6. external spaces
     - ➜ not full Iguana objects
     - ➜ serve as proxies for non-Iguana objects
- Access controlled by *capabilities*

Objects: Commonalities

- Objects have a unique name — *object ID* (OID)
  - ➜ OIDs are addresses in Iguana's SAS
  - ➜ only for memory sections does this address correspond to actual memory

Objects: Commonalities

- Objects have a unique name — *object ID* (OID)
  - ➜ OIDs are addresses in Iguana's SAS
  - ➜ only for memory sections does this address correspond to actual memory

- Objects have *methods* that can be invoked
  - ⋆ one method that exists for all objects: *destroy*
  - ⋆ each kind of object has a set of pre-defined methods

OBJECTS: COMMONALITIES

- Objects have a unique name — *object ID* (OID)
  - ➜ OIDs are addresses in Iguana's SAS
  - ➜ only for memory sections does this address correspond to actual memory
- Objects have *methods* that can be invoked
  - ⋆ one method that exists for all objects: *destroy*
  - ⋆ each kind of object has a set of pre-defined methods
- Objects are created by invoking constructor on a PD:
  - ➜ *kind*`_cap = `*pd*`->new_`*kind*`(`*args*`);`

Objects: Commonalities

- Objects have a unique name — *object ID* (OID)
  - ➜ OIDs are addresses in Iguana's SAS
  - ➜ only for memory sections does this address correspond to actual memory

- Objects have *methods* that can be invoked
  - ⋆ one method that exists for all objects: *destroy*
  - ⋆ each kind of object has a set of pre-defined methods

- Objects are created by invoking constructor on a PD:
  - ➜ *kind*`_cap = `*pd*`->new_`*kind*`(`*args*`);`

- Methods are grouped into *interfaces*
  - ⋆ interfaces also have unique IDs (IIDs) that are OID + interface number
  - ⋆ interfaces have capabilities
    - ➜ grant rights to invoke an interface's methods
  - ⋆ all pre-defined methods belong to separate interfaces
    - ➜ i.e., access is individually protected

DIFFERENCES TO MUNGI

- SAS name space extended to *all* objects
  - ➜ not just memory
  - ➜ supports unified access control and resource management

Differences to Mungi

- SAS name space extended to *all* objects
  - ➜ not just memory
  - ➜ supports unified access control and resource management
- Secondary storage not part of SAS
  - ➜ meant to run on 32-bit hardware
  - ➜ meant to run on embedded systems without disk

DIFFERENCES TO MUNGI

- SAS name space extended to *all* objects
  - ➜ not just memory
  - ➜ supports unified access control and resource management
- Secondary storage not part of SAS
  - ➜ meant to run on 32-bit hardware
  - ➜ meant to run on embedded systems without disk
- Provision for management of all resources (i.e. objects
  - ➜ only half worked out yet
  - ➜ one honours student working on details

DIFFERENCES TO MUNGI

- SAS name space extended to *all* objects
  - ➜ not just memory
  - ➜ supports unified access control and resource management
- Secondary storage not part of SAS
  - ➜ meant to run on 32-bit hardware
  - ➜ meant to run on embedded systems without disk
- Provision for management of all resources (i.e. objects
  - ➜ only half worked out yet
  - ➜ one honours student working on details
- Client-server model instead of migrating-threads model
  - ➜ more familiar to potential users
  - ➜ avoids some (as yet) unresolved resource management issues

DIFFERENCES TO MUNGI

- SAS name space extended to *all* objects
  - ➜ not just memory
  - ➜ supports unified access control and resource management
- Secondary storage not part of SAS
  - ➜ meant to run on 32-bit hardware
  - ➜ meant to run on embedded systems without disk
- Provision for management of all resources (i.e. objects
  - ➜ only half worked out yet
  - ➜ one honours student working on details
- Client-server model instead of migrating-threads model
  - ➜ more familiar to potential users
  - ➜ avoids some (as yet) unresolved resource management issues
- Aimed at commercial deployment ⇒ more conventional
  - ➜ Mungi is a pure research system
  - ➜ useful for trying out things

Iguana Capabilities

- A capability is a token that confers some access right(s)

IGUANA CAPABILITIES

- A capability is a token that confers some access right(s)

- Two kinds of capabilities in Iguana:

  - *master capability*
    - ➜ created when an object is created
    - ➜ confers rights on all methods of object
    - ➜ allows creation of further capabilities

  - *invocation capability*
    - ➜ created when an interface is created
    - ➜ confers right to invoke methods of a single interface

IGUANA CAPABILITIES

- A capability is a token that confers some access right(s)
- Two kinds of capabilities in Iguana:
  - ⋆ *master capability*
    - ➜ created when an object is created
    - ➜ confers rights on all methods of object
    - ➜ allows creation of further capabilities
  - ⋆ *invocation capability*
    - ➜ created when an interface is created
    - ➜ confers right to invoke methods of a single interface
- Capabilities are only active if stored in PD's *capability lists*
  - ➜ as in Mungi

Memory Sections

- Memory sections represent virtual memory
  - ⋆ allocation of a certain amount of virtual memory:

    mem_cap `=` *pd*`->new_mem(`*size*`);`

MEMORY SECTIONS

- Memory sections represent virtual memory
  - ⋆ allocation of a certain amount of virtual memory:

    mem_cap = *pd*->new_mem(*size*);

- Memory sections are the only objects that support user-defined methods

  ➜ others have pre-defined (standard) methods only

- Used to provide encapsulated services:
  - ⋆ service = memory (data) + server (thread) + methods

MEMORY SECTIONS...

- To create a service:
  - ⋆ register a server thread on memory section

    base`->new_server(`*thread_id*`);`

    ➜ *base* is the base address (OID) of the memory section
  - ⋆ register interfaces (user-defined methods)

    base `=` *iid*`->new_cap();`

    ➜ *iid* refers to number of new interface

MEMORY SECTIONS...

- To create a service:
  - ⋆ register a server thread on memory section

    ```
    base->new_server(thread_id);
    ```

    ➜ *base* is the base address (OID) of the memory section
  - ⋆ register interfaces (user-defined methods)

    ```
    base = iid->new_cap();
    ```

    ➜ *iid* refers to number of new interface
- Registering interfaces supports user-defined methods
  - ⋆ remember: each interface can have one or more methods

    ➜ interface number only interpreted by server

    ➜ similarly, the method number is an opcode delivered to the interface
  - ⋆ IIDs and method numbers allocated by system implementor

    ➜ part of the service's interface protocol

Memory Sections: Pseudo Methods

- Read (R), write (W), execute (X) are logically considered methods
  - ⋆ subjects them to same protection mechanisms as other methods
  - ⋆ no actual methods exist corresponding to those operations

Memory Sections: Pseudo Methods

- Read (R), write (W), execute (X) are logically considered methods
  - ⋆ subjects them to same protection mechanisms as other methods
  - ⋆ no actual methods exist corresponding to those operations
- Further pseudo-method is *clist* (C)
  - ⋆ allows holder to add an object to the Clist array

THREADS

- Iguana threads are essentially L4 threads:
  - threads within same PD operated on by plain L4 syscalls
    - ➜ correspond to local L4 threads (i.e., same L4 AS)
    - ➜ `ExchangeRegisters`, `IPC`

THREADS

- Iguana threads are essentially L4 threads:
  - threads within same PD operated on by plain L4 syscalls
    - correspond to local L4 threads (i.e., same L4 AS)
    - `ExchangeRegisters`, `IPC`
  - direct IPC to non-local threads is not allowed
    - use method invocations (corresponding to server thread)
    - presently not enforced by Iguana
    - requires enhancements to L4 (forthcoming API) to do efficiently
    - will provide attribute to ensure enforcement (at a cost)

THREADS

- Iguana threads are essentially L4 threads:
  - ⋆ threads within same PD operated on by plain L4 syscalls
    - ➜ correspond to local L4 threads (i.e., same L4 AS)
    - ➜ `ExchangeRegisters`, `IPC`
  - ⋆ direct IPC to non-local threads is not allowed
    - ➜ use method invocations (corresponding to server thread)
    - ➜ presently not enforced by Iguana
    - ➜ requires enhancements to L4 (forthcoming API) to do efficiently
    - ➜ will provide attribute to ensure enforcement (at a cost)
- Certain operations require privileges
  - ⋆ e.g. thread creation and deletion done by privileged L4 `ThreadControl()` call
- Done by Iguana on invocation of appropriate methods

Thread Operations

- Thread creation:

  thread_cap `=` *pd*`->new_thread(&`*l4_tid*`);`

  - returns two kinds of thread IDs
    - Iguana thread ID (*tid*), part of the *thread_cap*
      - ➜ used for protection and other Iguana-specific purposes
    - L4 thread ID (*l4_tid*)
      - ➜ used for L4 syscalls

Thread Operations

- Thread creation:

  thread_cap `= ` *pd*`->new_thread(&`*l4_tid*`);`

  ⋆ returns two kinds of thread IDs

  * ∗ Iguana thread ID (*tid*), part of the *thread_cap*
    - ➜ used for protection and other Iguana-specific purposes
  * ∗ L4 thread ID (*l4_tid*)
    - ➜ used for L4 syscalls

- New thread created *inactive*

  ⋆ can be activated by:

  - ➜ L4 syscall `ExchangeRegisters()` (local threads only)
  - ➜ Iguana method `tid->start(ip,sp)`

THREAD OPERATIONS...

- Obtain L4 thread ID

➜ `l4tid = tid->l4_tid();`

- Obtain own thread ID

➜ `tid = myself();`

- Obtain protection domain of thread

➜ `pd = tid->domain();`

- Obtain and modify scheduling parameters

➜ `tid->schedule_info(&info);`

Sessions

- Sessions reduce authentication overheads of repeated calls

SESSIONS

- Sessions reduce authentication overheads of repeated calls

- Prior to invoking methods on a service, must establish session

```
session = pd->new_session(server);
```

  - ⋆ establishes session between target PD and server

Sessions

- Sessions reduce authentication overheads of repeated calls
- Prior to invoking methods on a service, must establish session

```
session = pd->new_session(server);
```

  - ⋆ establishes session between target PD and server
  - ⋆ `server` is a PD ID
    - ➜ **Note:** This is likely to change

SESSIONS

- Sessions reduce authentication overheads of repeated calls
- Prior to invoking methods on a service, must establish session

```
session = pd->new_session(server);
```

  - ⋆ establishes session between target PD and server
  - ⋆ `server` is a PD ID
    - ➜ **Note:** This is likely to change
  - ⋆ Iguana informs the server by invoking its notification method

```
server->session_created(pd);
```

  - ⋆ Iguana notifies remaining partners if the session is destroyed

```
pd_or_server->session_destroyed(session);
```

IGUANA CAPABILITIES

- Iguana capabilities are user-level objects
  - ➜ *password capabilities*, consisting of OID and password

object ID	password

  - ➜ Length of password is configurable (normally $\geq$ 64 bits)

IGUANA CAPABILITIES

- Iguana capabilities are user-level objects
  - ➜ *password capabilities*, consisting of OID and password

object ID	password

  - ➜ Length of password is configurable (normally $\geq$ 64 bits)

- Same model as in Mungi
  - ➜ implicit presentation
  - ➜ two-level structure (Clists caps array, Clists)
  - ➜ same confinement approach

External Spaces

- External spaces are “raw” L4 address spaces
  - ➜ not part of Iguana SAS
- Provided to deal with restrictions of Iguana model
  - ➜ 32-bit address space may not be large enough to share between all protection domains
  - ➜ legacy support (e.g. strict `fork()` semantics) may require separate address spaces

External Spaces

- External spaces are “raw” L4 address spaces
  - ➜ not part of Iguana SAS
- Provided to deal with restrictions of Iguana model
  - ➜ 32-bit address space may not be large enough to share between all protection domains
  - ➜ legacy support (e.g. strict `fork()` semantics) may require separate address spaces
- External spaces come at a cost
  - ⋆ unable to make full use of fast address-space switching on ARM
  - ⋆ not well integrated with Iguana world
    - ➜ no fine-grained access control provided by Iguana capabilities
    - ➜ not allowed to communicate with any PD other than creator
      - ➜ not even with Iguana — cannot invoke methods
      - ➜ enforced via L4 redirectors

External Spaces — Operations

- Creation requires explicit specification of KIP and UTCB address

```
es = pd->new_es (kip, utcb_area);
```

- Thread creation also requires arguments similar to L4

```
l4tid = es->new_thread(pager,scheduler,starter,utcb);
```

HARDWARE ACCESS

- Device drivers need to access raw hardware features
- Iguana provides a (static) `hardware` object for this
  - ⋆ physical memory access:

    ```
    hardware->back_mem(adr, p_adr, caching);
    ```

    ➜ maps the memory section (`adr`) to the specified physical address with specified caching attributes
  - ⋆ interrupt association:

    ```
    hardware->register_interrupt(tid,irq);
    ```

    ➜ registers the specified thread as the handler of the specified interrupt

RESOURCE TOKENS

- Iguana’s resource management mechanism
- Note: presently this only exists conceptually
  - ➜ details of the model still need to be worked out
  - ➜ however, model is based on our experience with a similar model in Mungi

RESOURCE TOKENS

- Iguana's resource management mechanism
- Note: presently this only exists conceptually
  - ➜ details of the model still need to be worked out
  - ➜ however, model is based on our experience with a similar model in Mungi
- Basic idea: all resources have a price that must be paid by the user
- Model provides great flexibility for defining charging details
- Generalisation of Mungi bank accounts

WOMBAT: A PORTABLE LINUX SERVER ON L4

MOTIVATION

- Provide Linux API
- Support architectures other than x86
- Be easily maintainable
- Integrate with Iguana

LEGACY APIs ON L4

- Microkernel, like L4, provides mechanisms, not services
  - ➜ services implemented by user-level server
  - ➜ microkernel is OS agnostic
  - ➜ can, in principle, provide any OS API (OS “personality”)
  - ➜ can provide multiple OS APIs

LEGACY APIs ON L4

- Microkernel, like L4, provides mechanisms, not services
  - ➜ services implemented by user-level server
  - ➜ microkernel is OS agnostic
  - ➜ can, in principle, provide any OS API (OS “personality”)
  - ➜ can provide multiple OS APIs
- Multiple OS APIs are a classical *motivation* for microkernels

LEGACY APIs ON L4

- Microkernel, like L4, provides mechanisms, not services
  - ➜ services implemented by user-level server
  - ➜ microkernel is OS agnostic
  - ➜ can, in principle, provide any OS API (OS “personality”)
  - ➜ can provide multiple OS APIs

- Multiple OS APIs are a classical *motivation* for microkernels

- Simplest approach is to port an existing monolithic kernel
  - ➜ e.g., Unix, Linux
  - ➜ even several running concurrently

Legacy APIs on L4

- Microkernel, like L4, provides mechanisms, not services
  - ➜ services implemented by user-level server
  - ➜ microkernel is OS agnostic
  - ➜ can, in principle, provide any OS API (OS “personality”)
  - ➜ can provide multiple OS APIs
- Multiple OS APIs are a classical *motivation* for microkernels
- Simplest approach is to port an existing monolithic kernel
  - ➜ e.g., Unix, Linux
  - ➜ even several running concurrently
- Past experience poor:
  - ➜ Mach Unix server
  - ➜ IBM Workplace OS (Mach-based)

LEGACY APIs ON L4

- Microkernel, like L4, provides mechanisms, not services
  - ➜ services implemented by user-level server
  - ➜ microkernel is OS agnostic
  - ➜ can, in principle, provide any OS API (OS “personality”)
  - ➜ can provide multiple OS APIs
- Multiple OS APIs are a classical *motivation* for microkernels
- Simplest approach is to port an existing monolithic kernel
  - ➜ e.g., Unix, Linux
  - ➜ even several running concurrently
- Past experience poor:
  - ➜ Mach Unix server
  - ➜ IBM Workplace OS (Mach-based)
- Attributed to poor Mach IPC performance, large cache footprint

Porting Monolithic Kernels: Pros

- Code reuse: Existing services
- Code reuse: Legacy support
- Coder reuse
- Buzzword reuse

PORTING MONOLITHIC KERNELS: PROS

- Code reuse: Existing services

  ➜ can use existing code unchanged (file systems, network stacks, drivers, ...)
  ➜ dramatic reduction of cost of building new system
  ➜ potential reduction in maintenance cost (but...)

- Code reuse: Legacy support

- Coder reuse

- Buzzword reuse

PORTING MONOLITHIC KERNELS: PROS

- Code reuse: Existing services
  - ➜ can use existing code unchanged (file systems, network stacks, drivers, ...)
  - ➜ dramatic reduction of cost of building new system
  - ➜ potential reduction in maintenance cost (but...)
- Code reuse: Legacy support
  - ➜ can support huge number of existing apps (if sufficiently compatible)
  - ➜ dramatic reduction of cost of deploying new system
- Coder reuse
- Buzzword reuse

PORTING MONOLITHIC KERNELS: PROS

- Code reuse: Existing services
  - ➜ can use existing code unchanged (file systems, network stacks, drivers, ...)
  - ➜ dramatic reduction of cost of building new system
  - ➜ potential reduction in maintenance cost (but...)
- Code reuse: Legacy support
  - ➜ can support huge number of existing apps (if sufficiently compatible)
  - ➜ dramatic reduction of cost of deploying new system
- Coder reuse
  - ➜ can provide a familiar target for programmers
  - ➜ reduction of cost of maintaining system and new app development
- Buzzword reuse

PORTING MONOLITHIC KERNELS: PROS

- Code reuse: Existing services
  - ➜ can use existing code unchanged (file systems, network stacks, drivers, ...)
  - ➜ dramatic reduction of cost of building new system
  - ➜ potential reduction in maintenance cost (but...)
- Code reuse: Legacy support
  - ➜ can support huge number of existing apps (if sufficiently compatible)
  - ➜ dramatic reduction of cost of deploying new system
- Coder reuse
  - ➜ can provide a familiar target for programmers
  - ➜ reduction of cost of maintaining system and new app development
- Buzzword reuse
  - ➜ can jump on Linux bandwagon while really doing something more sensible...
  - ➜ good PR value ;-)

Porting Monolithic Kernels: Cons

- Performance
  - ⋆ Microkernel inherently adds overhead
    - ➜ microkernel-mediated rather than direct access to hardware
    - ➜ IPC to server (4 mode switches) rather than syscall (2 mode switches)

$\Rightarrow$ Legacy system on L4 will **always** be slower than native!

Porting Monolithic Kernels: Cons

- Performance
  - ⋆ Microkernel inherently adds overhead
    - ➜ microkernel-mediated rather than direct access to hardware
    - ➜ IPC to server (4 mode switches) rather than syscall (2 mode switches)

⇒ Legacy system on L4 will **always** be slower than native!

- ⋆ Question is: *How much slower???*

Porting Monolithic Kernels: Cons

- Miss out on most advantages of microkernels

PORTING MONOLITHIC KERNELS: CONS

- Miss out on most advantages of microkernels
  - ⋆ software-engineering advantages
    - ➜ microkernels can provide strongly encapsulated components with small interfaces
    - ➜ Linux approach is unmaintainable in the long run [Schach *et al*, Maintainability of the Linux Kernel, IEE Proceedings Software, **149** 18–23 (2002)]

PORTING MONOLITHIC KERNELS: CONS

- Miss out on most advantages of microkernels
  - ⋆ software-engineering advantages
    - ➜ microkernels can provide strongly encapsulated components with small interfaces
    - ➜ Linux approach is unmaintainable in the long run [Schach *et al*, Maintainability of the Linux Kernel, IEE Proceedings Software, **149** 18–23 (2002)]
  - ⋆ reduction of *trusted computing base*
    - ➜ trusting 2–3Mloc vs 10kloc kernel plus core services and drivers

PORTING MONOLITHIC KERNELS: CONS

- Miss out on most advantages of microkernels
  - ⋆ software-engineering advantages
    - ➜ microkernels can provide strongly encapsulated components with small interfaces
    - ➜ Linux approach is unmaintainable in the long run [Schach *et al*, Maintainability of the Linux Kernel, IEE Proceedings Software, **149** 18–23 (2002)]
  - ⋆ reduction of *trusted computing base*
    - ➜ trusting 2–3Mloc vs 10kloc kernel plus core services and drivers
  - ⋆ potential for formal verification of TCB?
    - ➜ might be feasible if it's small
    - ➜ definitely infeasible with something the size of Linux

PORTING MONOLITHIC KERNELS: CONS

- Miss out on most advantages of microkernels
  - ⋆ software-engineering advantages
    - ➜ microkernels can provide strongly encapsulated components with small interfaces
    - ➜ Linux approach is unmaintainable in the long run [Schach *et al*, Maintainability of the Linux Kernel, IEE Proceedings Software, **149** 18–23 (2002)]
  - ⋆ reduction of *trusted computing base*
    - ➜ trusting 2–3Mloc vs 10kloc kernel plus core services and drivers
  - ⋆ potential for formal verification of TCB?
    - ➜ might be feasible if it's small
    - ➜ definitely infeasible with something the size of Linux
- Main-stream systems have notoriously poor real-time support
  - ➜ typically unsuitable for hard real time

Hybrid Systems: Best of Two Worlds?

- Legacy server and native apps running side-by-side
  - ➜ code/coder/buzzword reuse
  - ➜ software-engineering advantages for critical/new parts
  - ➜ hard real-time apps not affected by legacy system

HYBRID SYSTEMS: BEST OF TWO WORLDS?

- Legacy server and native apps running side-by-side
  - ➜ code/coder/buzzword reuse
  - ➜ software-engineering advantages for critical/new parts
  - ➜ hard real-time apps not affected by legacy system

- How about performance?
  - ➜ L4 primitives much faster than Mach and others
  - ➜ fast enough?

L4 on Linux: History

L^4Linux

- First port of Linux kernel to L4
- Done for ix86 at Dresden [SOSP 97]
  - ⋆ modified architecture-specific part of Linux to use L4 syscalls
  - ⋆ binary compatibility: syscall redirection to Linux server
    - ➜ syscall exception delivered to L^4Linux via exception IPC
  - ⋆ better performance using modified libc
    - ➜ syscall trap replaced by L4 IPC to Linux server
  - ⋆ re-done for 2.2, 2.4, 2.6
    - ➜ L^4Linux implementation highly dependent on Linux internals
    - ➜ significant effort for forward porting (start from scratch each time)

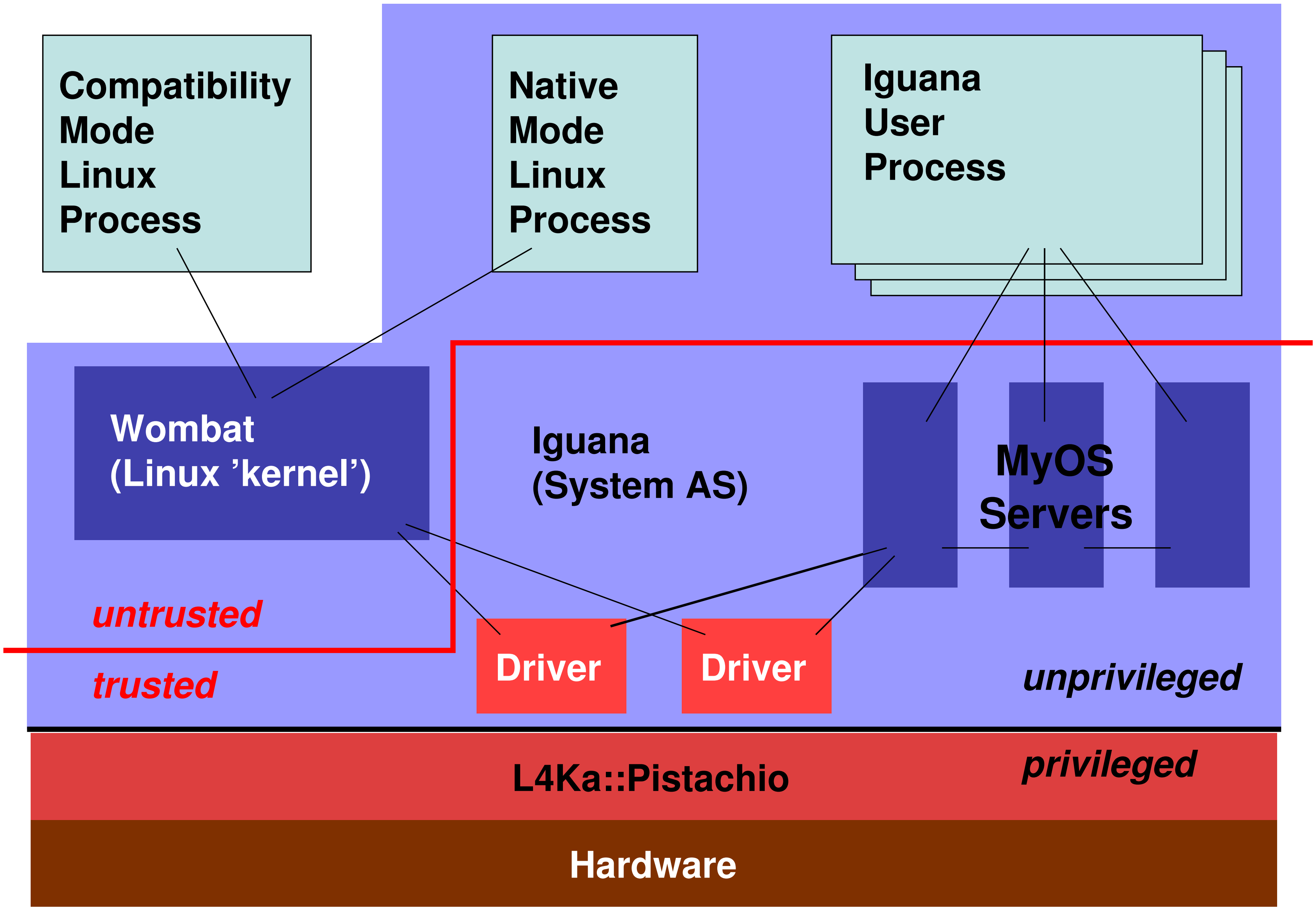
WOMBAT LINUX SERVER

- Done at NICTA 03–04
  - ➜ done from scratch, independent of Dresden work
- Portable between architectures
  - ➜ based on Iguana
  - ➜ presently runs on x86, ARM, MIPS64
  - ➜ MIPS64 supports 32-bit and 64-bit executables

WOMBAT LINUX SERVER

- Done at NICTA 03–04
  - ➜ done from scratch, independent of Dresden work
- Portable between architectures
  - ➜ based on Iguana
  - ➜ presently runs on x86, ARM, MIPS64
  - ➜ MIPS64 supports 32-bit and 64-bit executables
- Much reduced dependence on Linux internals
- Binary compatibility via syscall redirection
- Proper Linux scheduling of Linux apps
- Potential to achieve good performance on ARM

A SYSTEM RUNNING WOMBAT



WOMBAT IMPLEMENTATION

- Dresden L^4Linux approach:
  - ⋆ applied modifications to architecture-dependent part of kernel
    - ➜ very x86-specific

Wombat Implementation

- Dresden L^{4}Linux approach:
  - ⋆ applied modifications to architecture-dependent part of kernel
    - ➜ very x86-specific
- Wombat approach:
  - ⋆ introduced new `arch/l4/` in Linux source
    - ➜ kept as architecture-neutral as possible
    - ➜ architecture-dependent code in `arch/l4/sys-arm` etc

Wombat Implementation

- Dresden L^{4}Linux approach:
  - ⋆ applied modifications to architecture-dependent part of kernel
    - ➜ very x86-specific
- Wombat approach:
  - ⋆ introduced new `arch/l4/` in Linux source
    - ➜ kept as architecture-neutral as possible
    - ➜ architecture-dependent code in `arch/l4/sys-arm` etc
  - ⋆ otherwise only modified 3 Linux source files
    - ➜ two are one-line bug fixes (patches submitted)
    - ➜ third is `printk` (additional early debug output)

WOMBAT IMPLEMENTATION

- Dresden L^{4}Linux approach:
  - ⋆ applied modifications to architecture-dependent part of kernel
    - ➜ very x86-specific
- Wombat approach:
  - ⋆ introduced new `arch/l4/` in Linux source
    - ➜ kept as architecture-neutral as possible
    - ➜ architecture-dependent code in `arch/l4/sys-arm` etc
  - ⋆ otherwise only modified 3 Linux source files
    - ➜ two are one-line bug fixes (patches submitted)
    - ➜ third is `printk` (additional early debug output)
  - ⋆ tracked head revision of Linux from 2.5.x to 2.6.4
    - ➜ now at 2.9, will go forward soon

PAGE FAULTS AND EXCEPTIONS

- Wombat has a single server thread per CPU, like L^{4}Linux
  - is L4 pager and exception handler for Linux processes

Page Faults and Exceptions

- Wombat has a single server thread per CPU, like L4Linux
  - ⋆ is L4 pager and exception handler for Linux processes
  - ⋆ Linux “syscalls” are L4 exceptions

Page Faults and Exceptions

- Wombat has a single server thread per CPU, like L4Linux
  - ⋆ is L4 pager and exception handler for Linux processes
  - ⋆ Linux “syscalls” are L4 exceptions
  - ⋆ other exceptions are reflected back as Linux signals

Page Faults and Exceptions

- Wombat has a single server thread per CPU, like L^4Linux
  - ⋆ is L4 pager and exception handler for Linux processes
  - ⋆ Linux “syscalls” are L4 exceptions
  - ⋆ other exceptions are reflected back as Linux signals
- Have single runnable Linux user process per CPU, unlike L^4Linux
  - ⋆ differs from Dresden/Karlsruhe L^4Linux approaches
  - ⋆ helps to maintain Linux scheduling behaviour

System-Call Processing

- Access to application memory by Wombat:
  - ⋆ provide `copy_to`/`copy_from` functions
  - ⋆ look up page table and find page(s) in server's memory pool
  - ⋆ on MIPS could instead run Wombat in supervisor mode
    - ➜ would give direct access to user's address space

System-Call Processing

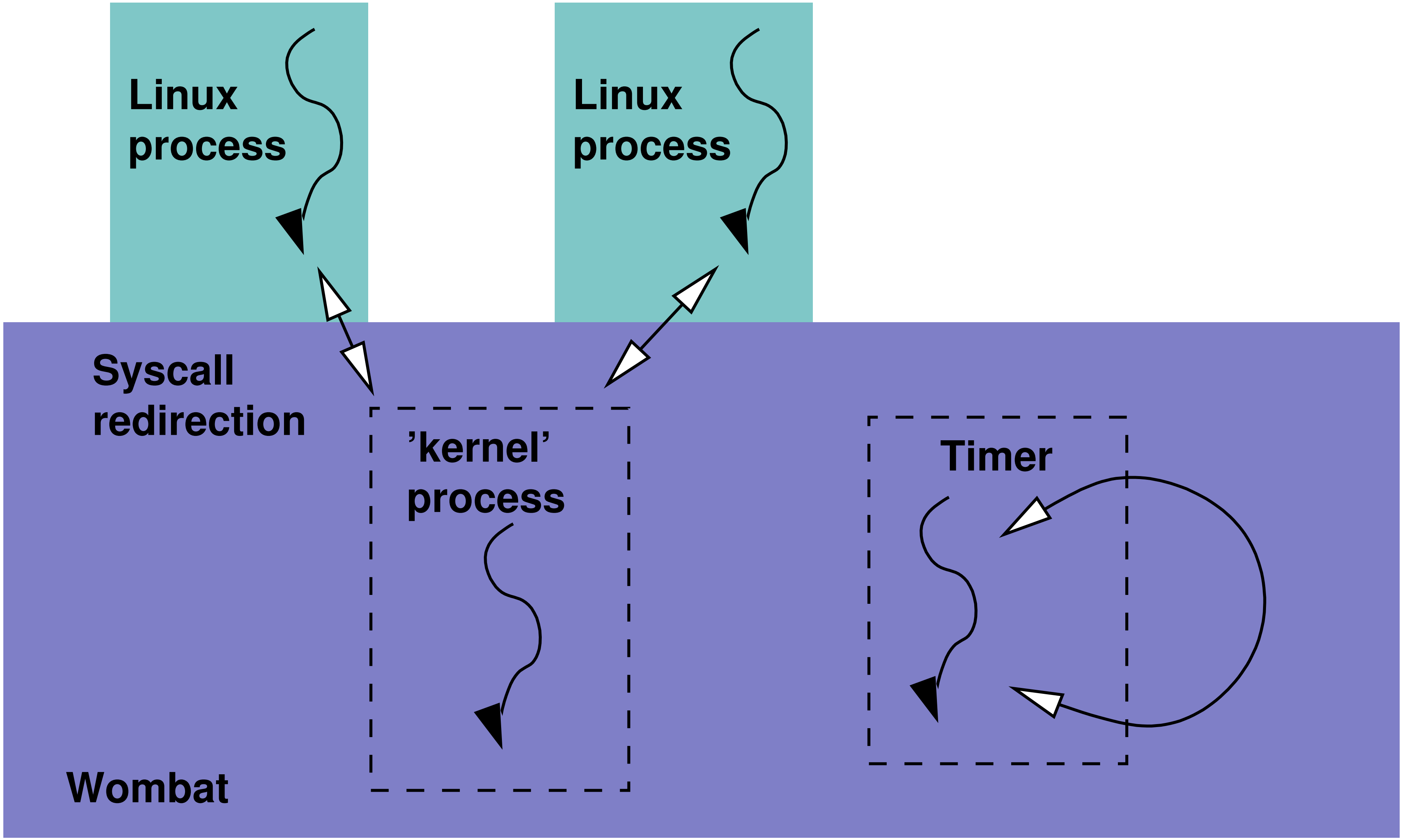
- Access to application memory by Wombat:
  - ⋆ provide `copy_to`/`copy_from` functions
  - ⋆ look up page table and find page(s) in server's memory pool
  - ⋆ on MIPS could instead run Wombat in supervisor mode
    - ➜ would give direct access to user's address space
- Context switching:
  - ⋆ Wombat maintains pointer to state of current Linux process
  - ⋆ On a process switch, switches state and updates pointer
    - ➜ on ARM, state is banked user registers

SYSTEM-CALL PROCESSING

- Access to application memory by Wombat:
  - ⋆ provide `copy_to`/`copy_from` functions
  - ⋆ look up page table and find page(s) in server's memory pool
  - ⋆ on MIPS could instead run Wombat in supervisor mode
    - ➜ would give direct access to user's address space
- Context switching:
  - ⋆ Wombat maintains pointer to state of current Linux process
  - ⋆ On a process switch, switches state and updates pointer
    - ➜ on ARM, state is banked user registers
- Signal processing is architecture dependent

➜ emulates Linux signal handling

LINUX PROCESSES



Wombat Scheduling

- Wombat has second thread: timer
  - ⋆ high-priority thread normally in timed wait
  - ⋆ maintains Linux time slice

Wombat Scheduling

- Wombat has second thread: timer
  - high-priority thread normally in timed wait
  - maintains Linux time slice
  - when woken:
    - ➜ checks whether same Linux process is still running
    - ➜ if so, sets *reschedule* flag
    - ➜ if flag already set then sets Linux process' *total quantum* to zero
    - ➜ forces preemption, and an IPC to the scheduler (the "kernel" thread)

WOMBAT SCHEDULING

- Wombat has second thread: timer
  - ⋆ high-priority thread normally in timed wait
  - ⋆ maintains Linux time slice
  - ⋆ when woken:
    - ➜ checks whether same Linux process is still running
    - ➜ if so, sets *reschedule* flag
    - ➜ if flag already set then sets Linux process' *total quantum* to zero
    - ➜ forces preemption, and an IPC to the scheduler (the "kernel" thread)
- Scheduling happens when Linux process enters kernel
  - ⋆ checks reschedule flag, if set:
    - ➜ invokes Linux scheduler
    - ➜ perform context switch, reset reschedule flag

Wombat Scheduling

- Actual scheduling decision made by *unmodified* Linux scheduler

➜ unlike L^4Linux

WOMBAT SCHEDULING

- Actual scheduling decision made by *unmodified* Linux scheduler
  - ➜ unlike L^{4}Linux
- Advantages of this approach:
  - ⋆ no changes required to (architecture-independent) scheduler
  - ⋆ Linux scheduling policy maintained (sort-of)
  - ⋆ no locking required other than what is already in Linux

Wombat Scheduling

- Actual scheduling decision made by *unmodified* Linux scheduler
  ➜ unlike L^4Linux
- Advantages of this approach:
  - ⋆ no changes required to (architecture-independent) scheduler
  - ⋆ Linux scheduling policy maintained (sort-of)
  - ⋆ no locking required other than what is already in Linux
- Disadvantages:
  - ⋆ Linux scheduling only approximately maintained
    - ➜ actual scheduling happens next time process enters kernel *after* time slice expired
    - ➜ processes run normally slightly longer than Linux time slice
    - ➜ CPU-bound process runs up to twice as long

WOMBAT SCHEDULING

- Actual scheduling decision made by *unmodified* Linux scheduler
  - ➜ unlike L4Linux
- Advantages of this approach:
  - ⋆ no changes required to (architecture-independent) scheduler
  - ⋆ Linux scheduling policy maintained (sort-of)
  - ⋆ no locking required other than what is already in Linux
- Disadvantages:
  - ⋆ Linux scheduling only approximately maintained
    - ➜ actual scheduling happens next time process enters kernel *after* time slice expired
    - ➜ processes run normally slightly longer than Linux time slice
    - ➜ CPU-bound process runs up to twice as long
    - ➜ Should be able to get exact Linux scheduling (but at a cost!)

LINUX IDLE TASK

- Linux's idle task executes `cpu_idle()`
  - ⋆ architecture-specific function
  - ⋆ in Wombat this IPCs to the "kernel" thread
  - ⋆ that one just sleeps until the next time tick or interrupt
  - ⋆ could just as well be made to go into a low-power mode

WOMBAT PERFORMANCE (PRELIMINARY)

			trampoline		IPC	
Benchmark	**Server**	**Linux**	**L4**	**ratio**	**L4**	**ratio**
`getpid()`	L4Linux	1.68	9.66	3.7	3.95	2.4
	Wombat	0.41	1.94	4.7		
2-proc ctxsw	L4Linux	7.0	18.2	2.6	16.2	2.3
	Wombat	4.03	9.66	2.4		
pipe	L4Linux	29.0	69.4	2.4	22.2	1.8
	Wombat	6.3	16.5	2.6		

- ➜ L4Linux 2.0 on 133MHz Pentium [SOSP 97], some improvements since
- ➜ Wombat 2.6 on Pentium-4 Xeon 2.66Ghz, *unoptimised*
  - ➜ overhead should be smaller on RISC processors

IGUANA DEVICE DRIVERS

- Drivers are Iguana services (data + server thread + methods)

IGUANA DEVICE DRIVERS

- Drivers are Iguana services (data + server thread + methods)

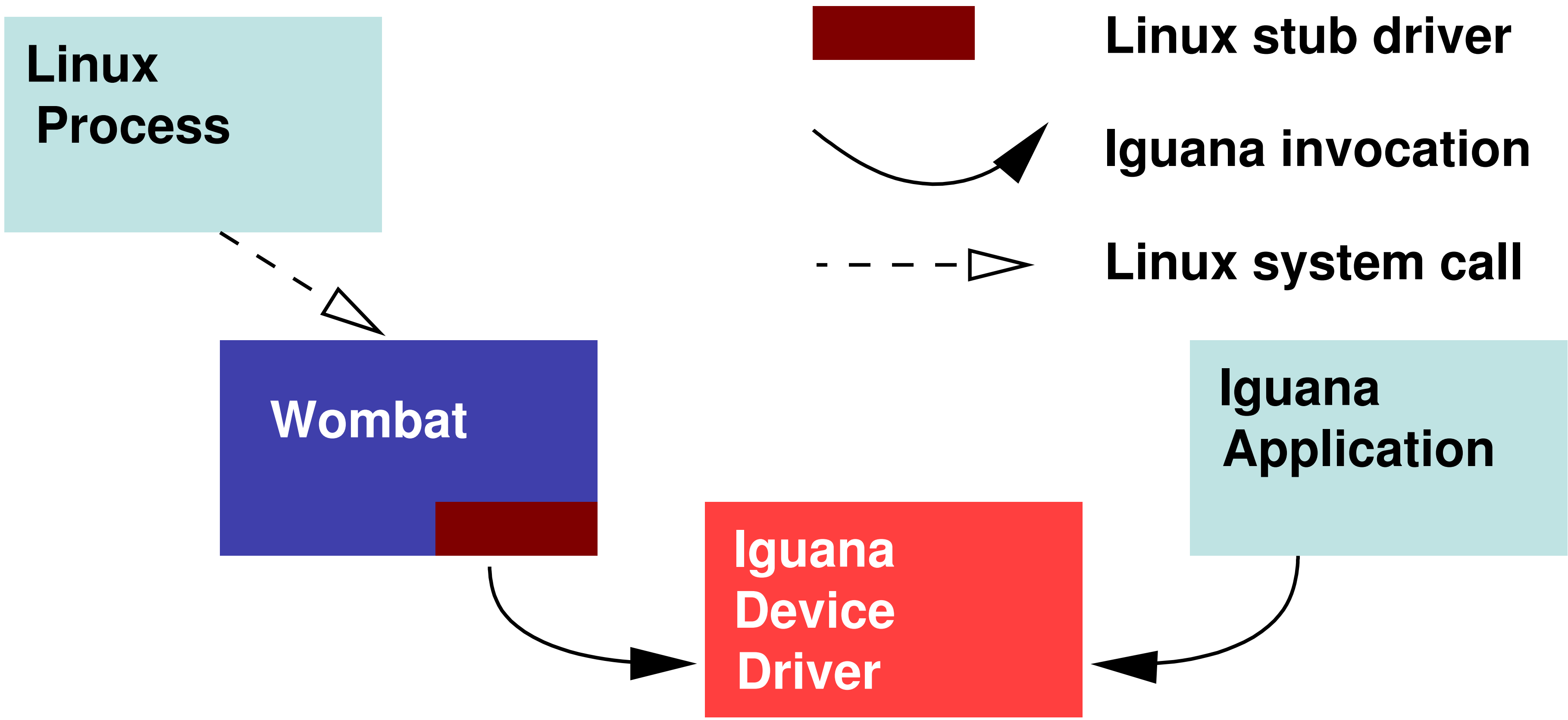
- At startup obtain access to hardware resources
  - ⋆ can be done first-come, first-served
  - ⋆ better: restrict allocation to privileged service
    - ➜ ... by restricting capabilities to the `hardware` object
    - ➜ privileged service can allocate device memory (using `hardware->back_mem`
    - ➜ can then hand capability for device-backed memory section to driver

- Driver is then able to operate the hardware

IGUANA DEVICE DRIVERS

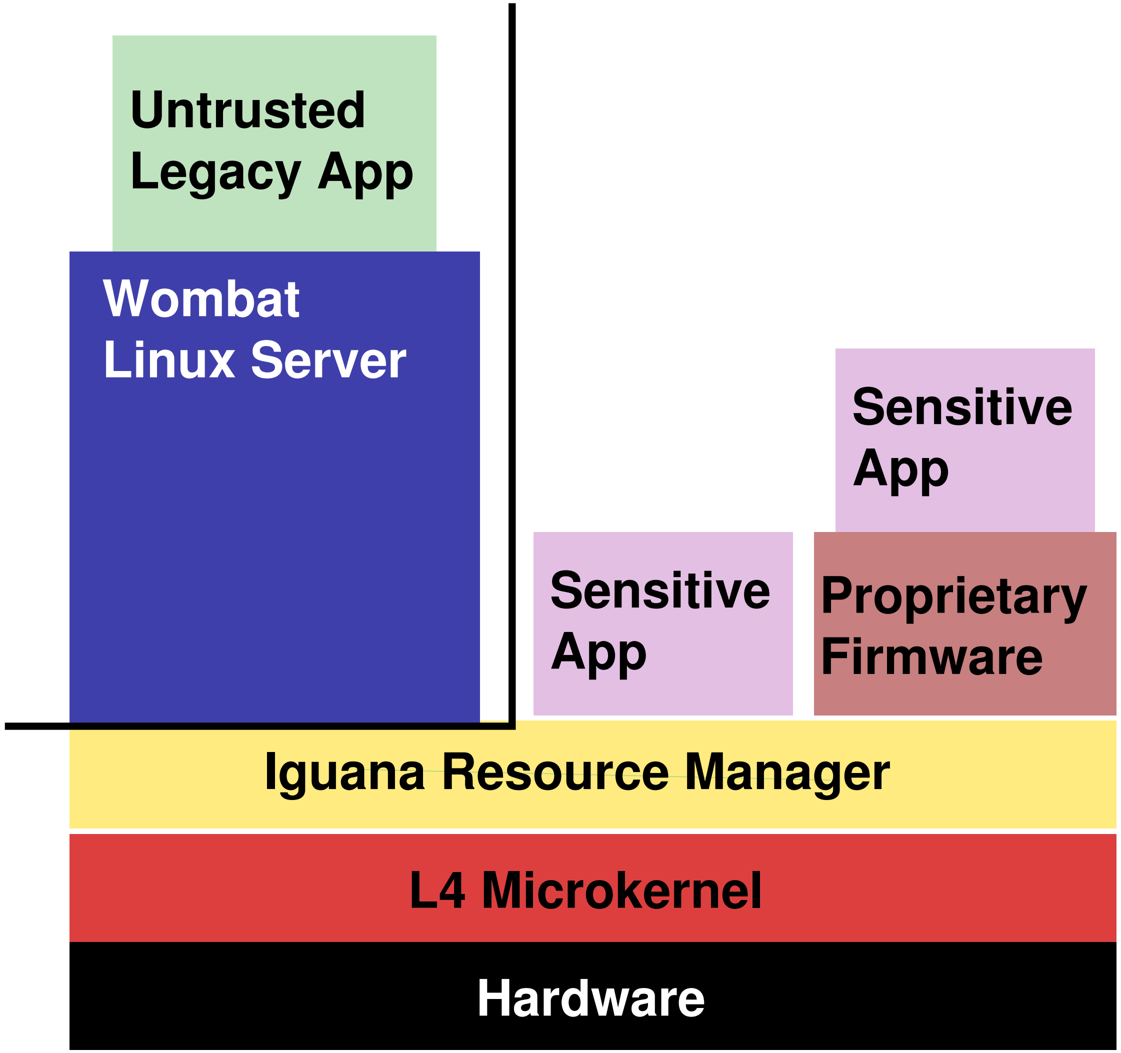
- Drivers are Iguana services (data + server thread + methods)
- At startup obtain access to hardware resources
  - ⋆ can be done first-come, first-served
  - ⋆ better: restrict allocation to privileged service
    - ➜ ... by restricting capabilities to the `hardware` object
    - ➜ privileged service can allocate device memory (using `hardware->back_mem`
    - ➜ can then hand capability for device-backed memory section to driver
- Driver is then able to operate the hardware
- Provides methods as per Iguana driver protocol

Sharing Iguana Device Drivers with Wombat



- Wombat has stub driver module
  - ➜ invokes Iguana driver
  - ➜ sample exists for console

TARGET SYSTEM ARCHITECTURE



Similar to Dresden DROPS system

Status: Iguana, Wombat and Driver Framework

- In use at (at least) two multinational companies
- Beta release Feb 05
- Proper release planned Nov 05
- Deployment in products planned for early next year

STATUS: IGUANA, WOMBAT AND DRIVER FRAMEWORK

- In use at (at least) two multinational companies
- Beta release Feb 05
- Proper release planned Nov 05
- Deployment in products planned for early next year
- Future work:
  - ➜ proper SMP implementation
  - ➜ support for heterogeneous multiprocessors (rSoC)
  - ➜ design and implementation of proper resource management

HIGH-PERFORMANCE USER-LEVEL DRIVERS

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

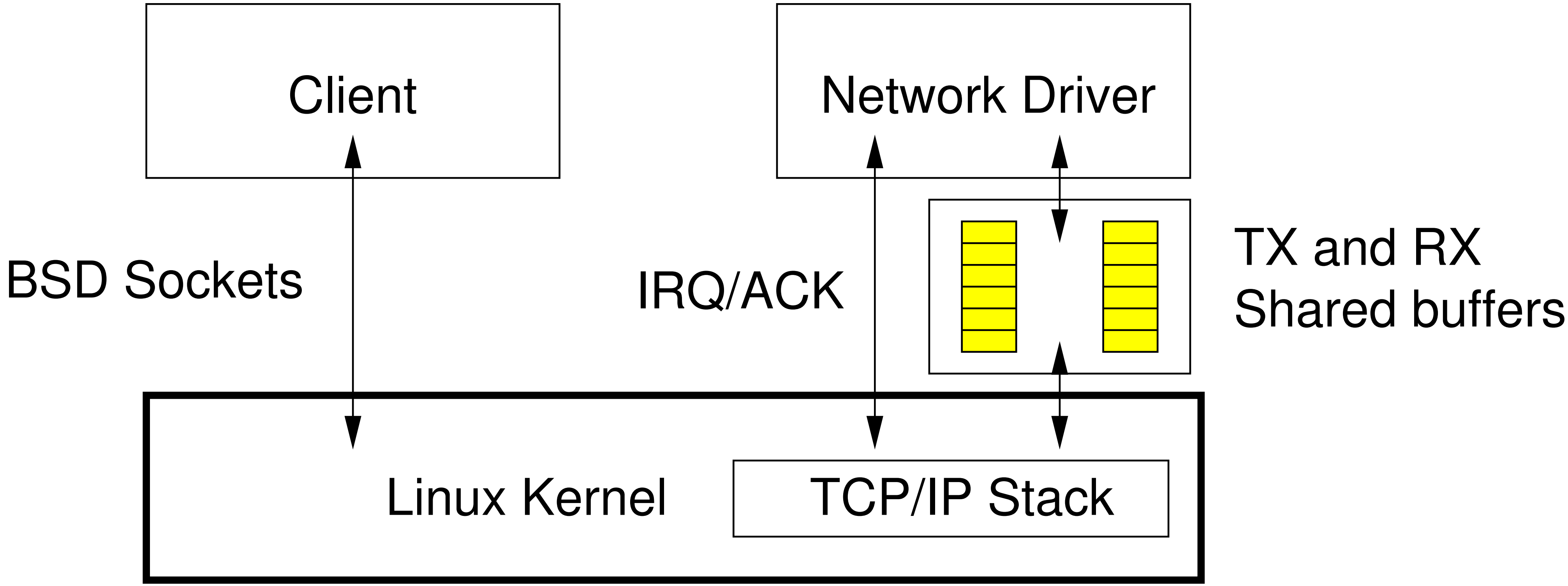
HIGH-PERFORMANCE USER-LEVEL DRIVERS

- 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

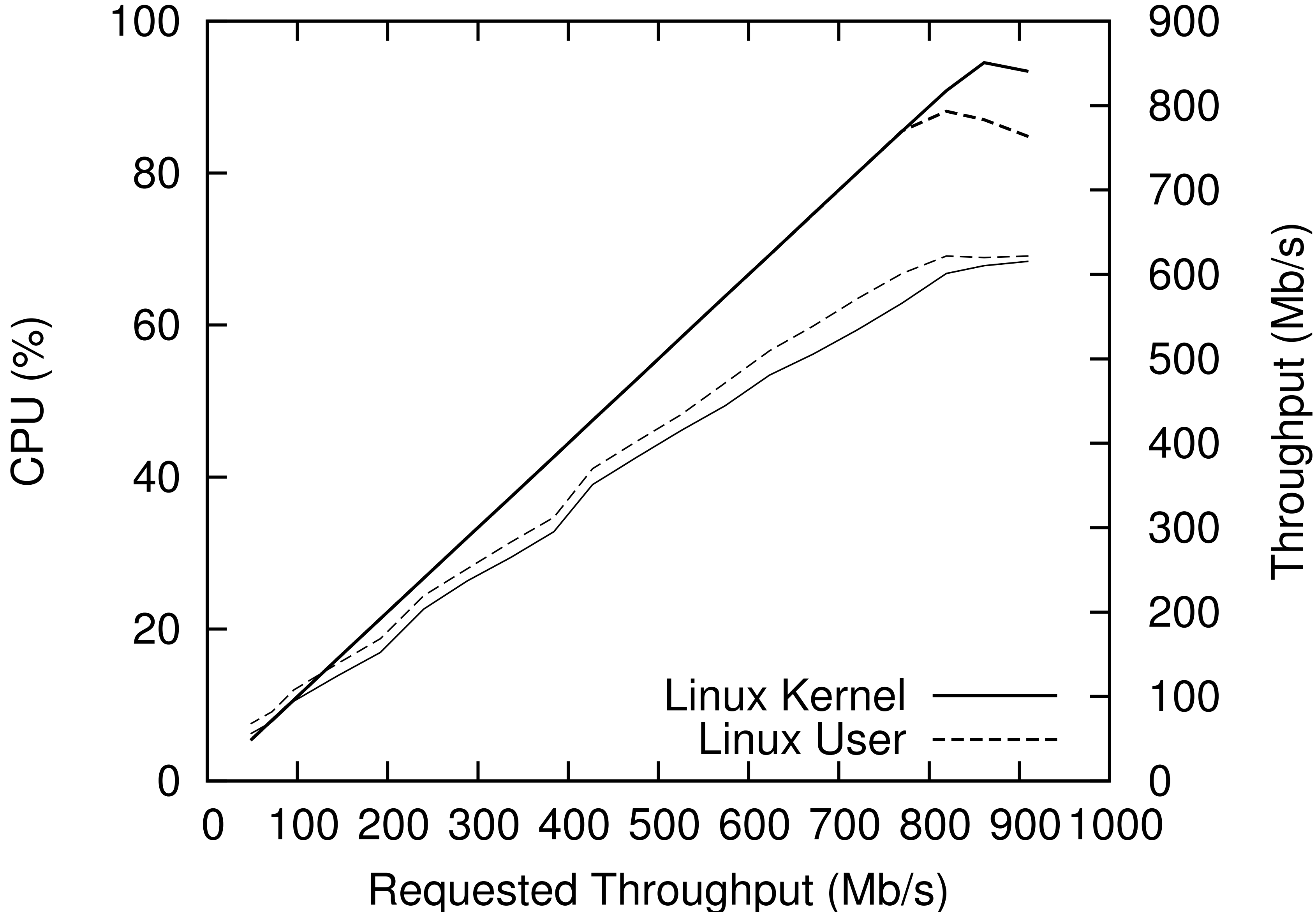
- L4 device drivers are *always* outside the kernel (at user level)
  - ⋆ Interrupts delivered to driver as IPC from kernel
- Potentially higher communication overhead
  - ⋆ past experience with user-level drivers: $\geq$50% performance degradation
- L4 IPC performance is very high
  - ⋆ with well-designed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

seL4: Secure Embedded L4

- Goal
  - ⋆ satisfy high security requirements
  - ⋆ support *certifiably secure systems*

seL4: Secure Embedded L4

- Goal
  - ⋆ satisfy high security requirements
  - ⋆ support *certifiably secure systems*
  - ➜ requires *proof of security properties*
  - ➜ present API is known not to support this

seL4: Secure Embedded L4

- Goal
  - ⋆ satisfy high security requirements
  - ⋆ support *certifiably secure systems*
    - ➜ requires *proof of security properties*
    - ➜ present API is known not to support this
- Issues:
  - ⋆ information flow control
    - ➜ V2: clans & chiefs: clumsy, inflexible, inefficient
    - ➜ V4: redirectors: still clumsy, often inefficient

seL4: Secure Embedded L4

- Goal
  - ⋆ satisfy high security requirements
  - ⋆ support *certifiably secure systems*
    - ➜ requires *proof of security properties*
    - ➜ present API is known not to support this
- Issues:
  - ⋆ information flow control
    - ➜ V2: clans & chiefs: clumsy, inflexible, inefficient
    - ➜ V4: redirectors: still clumsy, often inefficient
  - ⋆ IPC addressing via thread IDs reveals *global* names
    - ➜ violates *need to know*

seL4: Secure Embedded L4

- Goal
  - ⋆ satisfy high security requirements
  - ⋆ support *certifiably secure systems*
    - ➜ requires *proof of security properties*
    - ➜ present API is known not to support this
- Issues:
  - ⋆ information flow control
    - ➜ V2: clans & chiefs: clumsy, inflexible, inefficient
    - ➜ V4: redirectors: still clumsy, often inefficient
  - ⋆ IPC addressing via thread IDs reveals *global* names
    - ➜ violates *need to know*
  - ⋆ Primitive kernel memory management
    - ➜ allows denial-of-service attacks
    - ➜ reveals internal state

seL4: Secure Embedded L4

- Goal
  - ⋆ satisfy high security requirements
  - ⋆ support *certifiably secure systems*
    - ➜ requires *proof of security properties*
    - ➜ present API is known not to support this
- Issues:
  - ⋆ information flow control
    - ➜ V2: clans & chiefs: clumsy, inflexible, inefficient
    - ➜ V4: redirectors: still clumsy, often inefficient
  - ⋆ IPC addressing via thread IDs reveals *global* names
    - ➜ violates *need to know*
  - ⋆ Primitive kernel memory management
    - ➜ allows denial-of-service attacks
    - ➜ reveals internal state
- Collaboration with Dresden, Johns Hopkins (Jonathan Shapiro)
  - ➜ Kevin Elphinstone, 1 staff, students

Formal Verification of Microkernel

- First (and most significant) step to making the TCB *trustworthy*!

Formal Verification of Microkernel

- First (and most significant) step to making the TCB *trustworthy*!

- Involves three aspects:

1. develop a formal specification of the kernel API
   - ➜ seL4
2. prove safety properties about the API
3. prove that implementation is correct
   - ➜ never before been done for any protected kernel!
   - ➜ L4 (≈10,000 loc) is at limit of tractability
   - ➜ expect seL4 kernel to be 5000–7000 loc

Formal Verification of Microkernel

- First (and most significant) step to making the TCB *trustworthy*!
- Involves three aspects:
  1. develop a formal specification of the kernel API
     ➜ seL4
  2. prove safety properties about the API
  3. prove that implementation is correct
     ➜ never before been done for any protected kernel!
     ➜ L4 (≈10,000 loc) is at limit of tractability
     ➜ expect seL4 kernel to be 5000–7000 loc
- Ultimate goal: Verify a *real* kernel (L4) that is used in production
  ➜ collaboration with NICTA Formal Methods Program

Formal Verification of Microkernel

- First (and most significant) step to making the TCB *trustworthy*!
- Involves three aspects:

1. develop a formal specification of the kernel API
   - ➜ seL4
2. prove safety properties about the API
3. prove that implementation is correct
   - ➜ never before been done for any protected kernel!
   - ➜ L4 (≈10,000 loc) is at limit of tractability
   - ➜ expect seL4 kernel to be 5000–7000 loc

- Ultimate goal: Verify a *real* kernel (L4) that is used in production
  - ➜ collaboration with NICTA Formal Methods Program
- Reduce risk: pilot project

L4 VERIFICATION PILOT PROJECT

- Purpose:
  - test tools and get experience with them
  - feasibility study
  - bridge cultural gap between theorists and kernel hackers
  - develop project plan for full-scale verification project

L4 VERIFICATION PILOT PROJECT

- Purpose:
  - ⋆ test tools and get experience with them
  - ⋆ feasibility study
  - ⋆ bridge cultural gap between theorists and kernel hackers
  - ⋆ develop project plan for full-scale verification project
- Operates on a “slice” of the API (address-space management)
  - ⋆ done formal model of (simplified) API slice
  - ⋆ proved some properties (absence of loops in mappings)
  - ⋆ refined specification down to source code
    - ➜ page table operations and simplified mapping database

L4 Verification Project

- Timeline:
  - commenced April 2005
  - 3 years
- Resources: $\approx$ 20 person years

L4 Verification Project

- Timeline:
  - ⋆ commenced April 2005
  - ⋆ 3 years
- Resources: $\approx$ 20 person years
  - ⋆ presently:
    - ➜ 2 FTE researchers
    - ➜ 1 PhD student
    - ➜ 1 honours
  - ⋆ definitely looking for students!

L4 Verification Project

- Timeline:
  - ⋆ commenced April 2005
  - ⋆ 3 years
- Resources: ≈ 20 person years
  - ⋆ presently:
    - ➜ 2 FTE researchers
    - ➜ 1 PhD student
    - ➜ 1 honours
  - ⋆ definitely looking for students!
  - ⋆ desired background:
    - ➜ good systems background and aptitude for maths, or
    - ➜ maths background and aptitude for systems

PROJECT: L4 WCET ANALYSIS

- Hard real-time systems need guarantees of worst-case latencies
  - ⋆ Need *worst-case execution time* (WCET) of kernel operations

Project: L4 WCET Analysis

- Hard real-time systems need guarantees of worst-case latencies
  - Need *worst-case execution time* (WCET) of kernel operations
- Usually done:
  - by code inspection
    ➜ unfeasible for non-trivial kernel
  - experimentally
    ➜ unreliable: cannot guarantee 100% coverage!

Project: L4 WCET Analysis

- Hard real-time systems need guarantees of worst-case latencies
  - ⋆ Need *worst-case execution time* (WCET) of kernel operations
- Usually done:
  - ⋆ by code inspection
    - ➜ unfeasible for non-trivial kernel
  - ⋆ experimentally
    - ➜ unreliable: cannot guarantee 100% coverage!
  - ⋆ by educated guess
    - ➜ highly unreliable (RTLinux got it wrong!)
    - ➜ typically leads to vastly pessimistic latencies

Project: L4 WCET Analysis

- Hard real-time systems need guarantees of worst-case latencies
  - ⋆ Need *worst-case execution time* (WCET) of kernel operations
- Usually done:
  - ⋆ by code inspection
    ➜ unfeasible for non-trivial kernel
  - ⋆ experimentally
    ➜ unreliable: cannot guarantee 100% coverage!
  - ⋆ by educated guess
    ➜ highly unreliable (RTLinux got it wrong!)
    ➜ typically leads to vastly pessimistic latencies
- Need systematic execution-time analysis of complete kernel

➜ never been done for non-trivial kernel!

PROJECT: L4 WCET ANALYSIS

- Hard real-time systems need guarantees of worst-case latencies
  - ⋆ Need *worst-case execution time* (WCET) of kernel operations
- Usually done:
  - ⋆ by code inspection
    - ➜ unfeasible for non-trivial kernel
  - ⋆ experimentally
    - ➜ unreliable: cannot guarantee 100% coverage!
  - ⋆ by educated guess
    - ➜ highly unreliable (RTLinux got it wrong!)
    - ➜ typically leads to vastly pessimistic latencies
- Need systematic execution-time analysis of complete kernel

➜ never been done for non-trivial kernel!

➜ project leader Stefan Petters done similar (non-kernel) work

➜ looking for students!

Component-Based Software Architecture

- Goal: component architecture that is appropriate for microkernel

➜ based on Iguana

Component-Based Software Architecture

- Goal: component architecture that is appropriate for microkernel
  ➜ based on Iguana
- To support:
  - ⋆ effective software-engineering techniques
  - ⋆ robustness by encapsulation without destroying performance
  - ⋆ hot-swapping / hot upgrading of components
  - ⋆ real-time (WCET) analysis of system
  - ⋆ ultimately, formal verification of whole system

COMPONENT-BASED SOFTWARE ARCHITECTURE

- Goal: component architecture that is appropriate for microkernel
  - ➜ based on Iguana
- To support:
  - ⋆ effective software-engineering techniques
  - ⋆ robustness by encapsulation without destroying performance
  - ⋆ hot-swapping / hot upgrading of components
  - ⋆ real-time (WCET) analysis of system
  - ⋆ ultimately, formal verification of whole system
- Collaboration with NICTA Empirical Software Engineering Program
  - ➜ 3 staff, several PhD and U/G students, visitors

SECURE MOBILE CODE

- Combines language and OS techniques for secure execution
- Code may be:
  - ➜ accompanied by a proof certificate establishing its trustworthiness
  - ➜ otherwise sandboxed execution
- Lead by Manuel Chakravarty
  - ⋆ presently working on theoretical foundations
  - ⋆ later to be integrated with Iguana
  - ⋆ requires *configurable protection domains*

GOANNA: STATIC ANALYSIS OF SYSTEMS SOFTWARE

- Model-checking-based tools looking for bugs in kernel & libs
- Collaboration with NICTA Formal Methods Program
- Pilot Project April 05 – March 06
  - ➜ examine state of the art
  - ➜ evaluate existing tools
  - ➜ analysis framework
  - ➜ get experience with model-checking of systems code
  - ➜ develop plan for follow-on project

PROJECT: OS-FRIENDLINESS OF ARCHITECTURES

- Evaluate hardware-dictated OS overheads and trends
- Critical operations
  - ➜ kernel entry and exit
  - ➜ TLB refill
  - ➜ context switch
  - ➜ synchronization primitives
- Hardware characteristics
  - ➜ instruction-set architecture
  - ➜ clock speed
  - ➜ pipeline depth
  - ➜ issue width
- Aim: paper in architecture conference

PROJECT: KERNEL IMPLEMENTATION TRADE OFFS

- Idea: revisit some of Jochen's tricks
  - ➜ virtually- vs physically-addressed TCBs
  - ➜ per-thread vs per-CPU kernel stack
  - ➜ exception IPC vs exception callback
- Question: how much influence of architecture?
- Aim: Conference paper

Project: L4Darwin

- Aim: Mac OS done right
  - ➜ remove Mach
  - ➜ substitute L4
  - ➜ de-privilege Darwin
  - ➜ de-privilege IOkit
  - ➜ kick-ass performance

PROJECT: L4DARWIN

- Aim: Mac OS done right
  - ➜ remove Mach
  - ➜ substitute L4
  - ➜ de-privilege Darwin
  - ➜ de-privilege IOkit
  - ➜ kick-ass performance
- Aim: exercise for the reader

GELATO: LINUX PERFORMANCE AND SCALABILITY

- VM system performance and scalability
  - ➜ superpage support (particularly Itanium)
  - ➜ NUMA support
  - ➜ guarded page tables for large address spaces
- TCP/IP and NFS throughput and scalability
- NUMA-aware scheduling
- Aims: support
  - ⋆ 1000's of CPUs in a NUMA system
  - ⋆ 1000's of nodes in a cluster
  - ⋆ terabytes of RAM
  - ⋆ support commercial workloads
- Funded by HP and ARC, supported by SGI

Other projects

- Pre-virtualisation
  - ➜ semi-automated high-performance virtualisation
  - ➜ covered in Week 6
- Hot swapping/upgrading of system components
  - ➜ collaboration with IBM research

OTHER PROJECTS

- Pre-virtualisation
  - ➜ semi-automated high-performance virtualisation
  - ➜ covered in Week 6
- Hot swapping/upgrading of system components
  - ➜ collaboration with IBM research
- Distributed embedded systems
  - ➜ e.g. cars
  - ➜ pilot project SUNswift

Other projects...

- Secure GUI
  - ➜ window system that ensures isolation of clients

OTHER PROJECTS...

- Secure GUI
  - ➜ window system that ensures isolation of clients
- JVM on L4

Other projects...

- Secure GUI
  - ➜ window system that ensures isolation of clients
- JVM on L4
- API emulation
  - ➜ support Symbian OS, QNX, ...

OTHER PROJECTS...

- Secure GUI
  - ➜ window system that ensures isolation of clients
- JVM on L4
- API emulation
  - ➜ support Symbian OS, QNX, ...
- Kernel ports
  - ➜ Blackfin, Hitachi, SPARC, Motorola

OTHER PROJECTS...

- Secure GUI
  - ➜ window system that ensures isolation of clients
- JVM on L4
- API emulation
  - ➜ support Symbian OS, QNX, ...
- Kernel ports
  - ➜ Blackfin, Hitachi, SPARC, Motorola
- Simulators
  - ➜ tools for OS development and architecture research
  - ➜ ARM, PowerPC, ...

SUMMARY: OS RESEARCH AT UNSW/NICTA

- Improve reliability and trustworthiness of embedded systems
  - ⋆ make a *real* impact on the practice of embedded systems R&D

SUMMARY: OS RESEARCH AT UNSW/NICTA

- Improve reliability and trustworthiness of embedded systems
  - ⋆ make a *real* impact on the practice of embedded systems R&D
- Comprehensive approach used on using microkernel technology
- Combination of formal and practical approaches
- Leverage relevant competencies available within NICTA

SUMMARY: OS RESEARCH AT UNSW/NICTA

- Improve reliability and trustworthiness of embedded systems
  - ⋆ make a *real* impact on the practice of embedded systems R&D
- Comprehensive approach used on using microkernel technology
- Combination of formal and practical approaches
- Leverage relevant competencies available within NICTA
- Ample opportunities to participate in cutting-edge research
  - ➜ summer projects
  - ➜ honours theses
  - ➜ PhD theses
  - ➜ employment