Microkernel-based Systems Summer School 2013: Genode OS Framework

Norman Feske
<norman.feske@genode-labs.com>

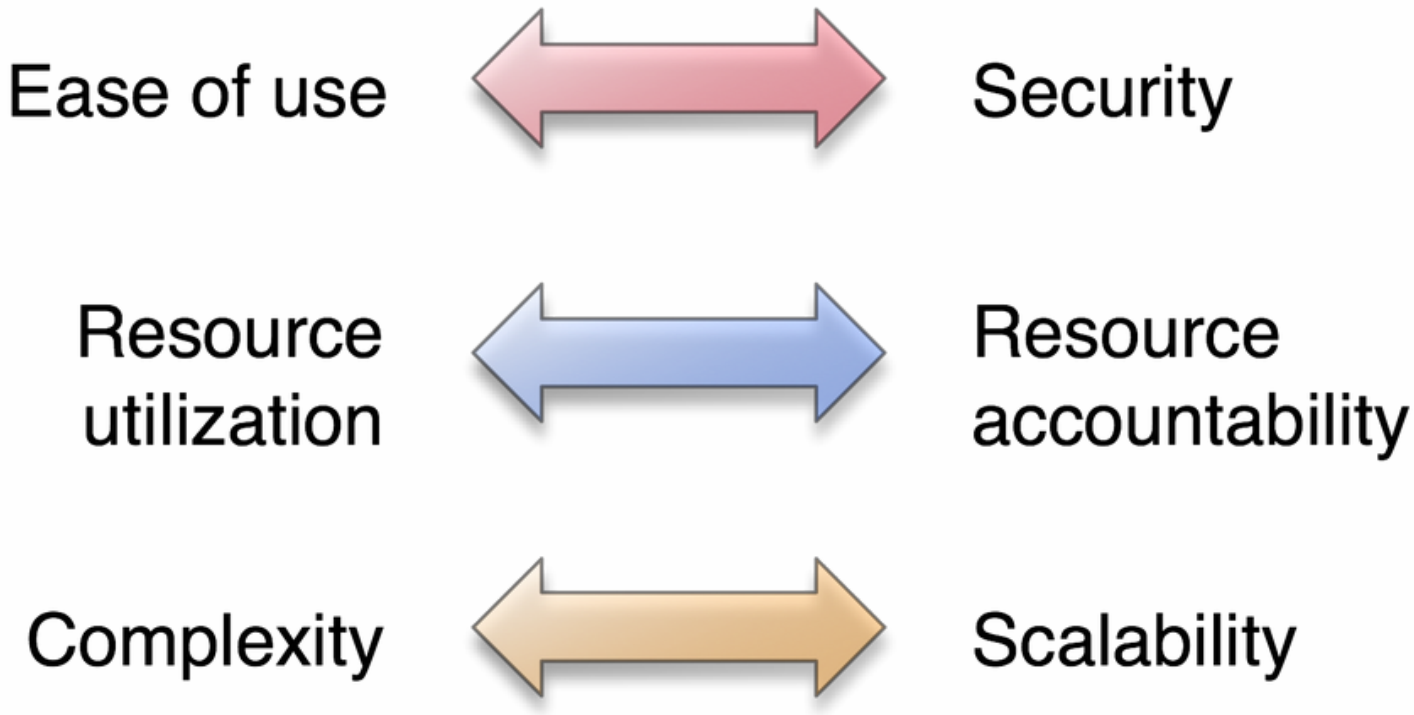
Outline

1. Why do we need another operating system?
2. Genode entering the picture
3. Architectural Principles
4. Core - the root of the process tree
5. Inter-process communication
6. Classification of components
7. Kernelization example
8. Components overview

Outline

1. Why do we need another operating system?
2. Genode entering the picture
3. Architectural Principles
4. Core - the root of the process tree
5. Inter-process communication
6. Classification of components
7. Kernelization example
8. Components overview

Myths


Ease of use
Security
Resource utilization
Resource accountability
Complexity
Scalability

Problem: Complexity

Today's commodity OSes Exceedingly complex trusted computing base (TCB)

TCB of an application on Linux:

- Kernel + loaded kernel modules
- Daemons
- X Server + window manager
- Desktop environment
- All running processes of the user

→ **User credentials are exposed to millions of lines of code**

Problem: Complexity (II)

Implications:

- High likelihood for bugs (need for frequent security updates)
- Huge attack surface for directed attacks
- Zero-day exploits

Problem: Global names

- Many examples on traditional systems
  - UIDs, PIDs
  - network interface names
  - port numbers
  - device nodes
  - ...
- Leak information
- Name is a potential attack vector (ambient authority)

Problem: Resource management

- Pretension of unlimited resources
- Lack of accounting

  → Largely indeterministic behavior

  → Need for complex heuristics, schedulers

```
Jul 24 12:58:30 neo kernel: [72454.482259] cupsd invoked oom-killer: gfp_mask=0x201da, order=0, oom_adj=0, oom_score_adj=0
Jul 24 12:58:30 neo kernel: [72454.482264] cupsd cpuset=/ mems_allowed=0
Jul 24 12:58:30 neo kernel: [72454.482268] Pid: 1416, comm: cupsd Tainted: G        WC  3.0.0-22-generic #36-Ubuntu
Jul 24 12:58:30 neo kernel: [72454.482270] Call Trace:
Jul 24 12:58:30 neo kernel: [72454.482279]  [<ffffffff810b5c8d>] ? cpuset_print_task_mems_allowed+0x9d/0xb0
Jul 24 12:58:30 neo kernel: [72454.482286]  [<ffffffff8110df91>] dump_header+0x91/0xe0
Jul 24 12:58:30 neo kernel: [72454.482289]  [<ffffffff8110e2f5>] oom_kill_process+0x85/0xb0
Jul 24 12:58:30 neo kernel: [72454.482293]  [<ffffffff8110e69a>] out_of_memory+0xfa/0x250
Jul 24 12:58:30 neo kernel: [72454.482298]  [<ffffffff81113f3f>] __alloc_pages_nodemask+0x80f/0x820
Jul 24 12:58:30 neo kernel: [72454.482304]  [<ffffffff8120a1a0>] ? noalloc_get_block_write+0x30/0x30
Jul 24 12:58:30 neo kernel: [72454.482311]  [<ffffffff8114a0a3>] alloc_pages_current+0xa3/0x110
Jul 24 12:58:30 neo kernel: [72454.482314]  [<ffffffff8110ab4f>] __page_cache_alloc+0x8f/0xa0
Jul 24 12:58:30 neo kernel: [72454.482318]  [<ffffffff8110afae>] ? find_get_page+0x1e/0x90
Jul 24 12:58:30 neo kernel: [72454.482321]  [<ffffffff8110cea4>] filemap_fault+0x234/0x3e0
Jul 24 12:58:30 neo kernel: [72454.482326]  [<ffffffff8115f07b>] ? mem_cgroup_update_page_stat+0x2b/0x110
Jul 24 12:58:30 neo kernel: [72454.482330]  [<ffffffff8112caf4>] __do_fault+0x54/0x510
Jul 24 12:58:30 neo kernel: [72454.482334]  [<ffffffff8113021a>] handle_pte_fault+0xfa/0x210
Jul 24 12:58:30 neo kernel: [72454.482337]  [<ffffffff811306e8>] handle_mm_fault+0x1f8/0x350
Jul 24 12:58:30 neo kernel: [72454.482344]  [<ffffffff815f8913>] do_page_fault+0x153/0x530
Jul 24 12:58:30 neo kernel: [72454.482350]  [<ffffffff81011069>] ? read_tsc+0x9/0x20
Jul 24 12:58:30 neo kernel: [72454.482355]  [<ffffffff8108cd2d>] ? ktime_get_ts+0xad/0xe0
Jul 24 12:58:30 neo kernel: [72454.482361]  [<ffffffff8117bb6a>] ? poll_select_set_timeout+0x7a/0x90
Jul 24 12:58:30 neo kernel: [72454.482365]  [<ffffffff815f5615>] page_fault+0x25/0x30
Jul 24 12:58:30 neo kernel: [72454.493363] Out of memory: Kill process 22727 (oom) score 691 or sacrifice child
Jul 24 12:58:30 neo kernel: [72454.493367] Killed process 22727 (oom) total-vm:2702616kB, anon-rss:2701332kB, file-rss:172kB
```

Key technologies

- Microkernels
- Decomponentization, kernelization
- Capability-based security
- Virtualization

Tricky questions

How to...

- ...build a system without global names?
- ...trade between parties that do not know each other?
- ...reclaim kidnapped goods from an alien? (without violence)
- ...deal with distributed access-control policies?
- ...transparently monitor communication?
- ...recycle a subsystem without knowing its internal structure?

Even more tricky questions

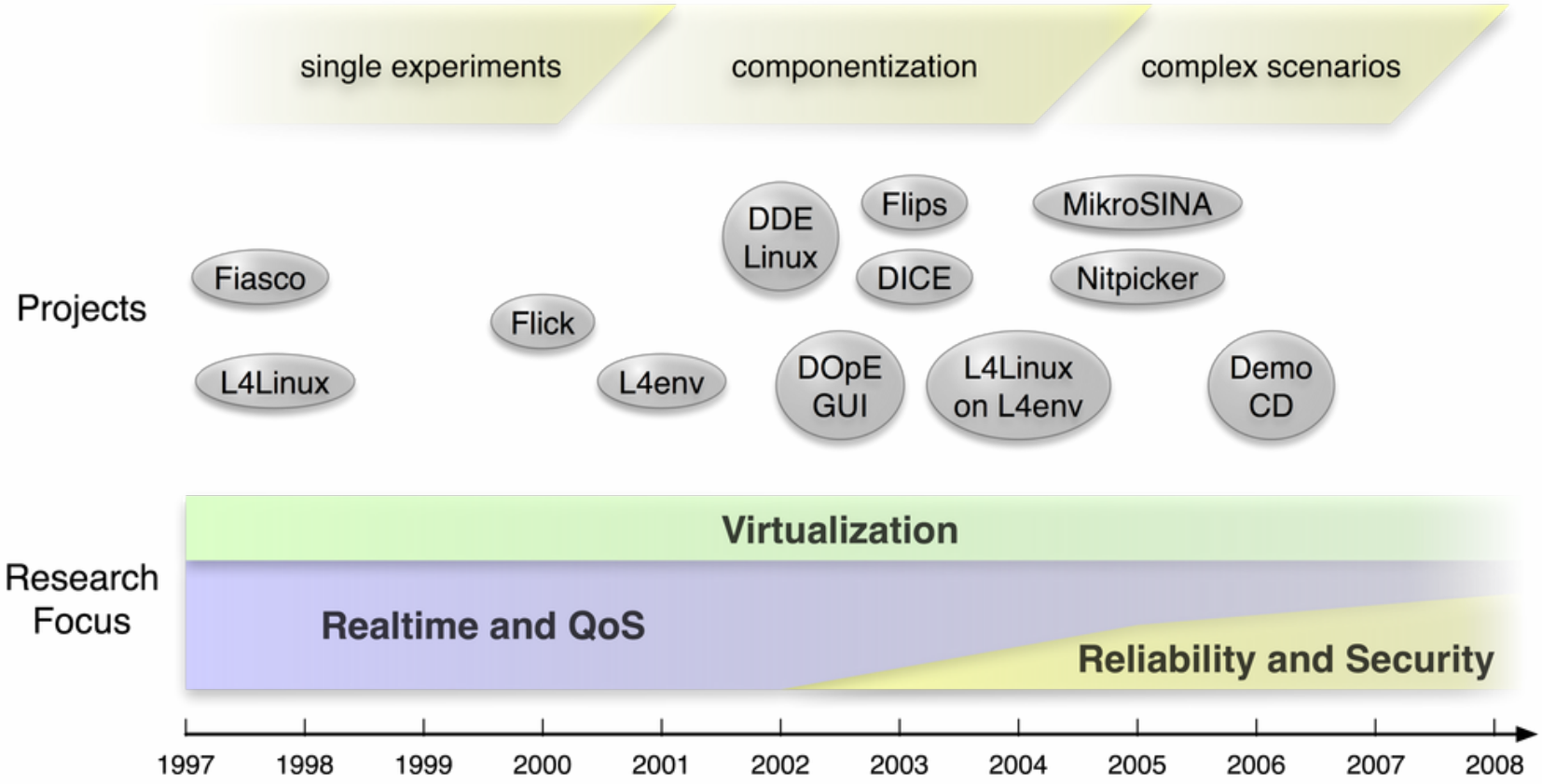
How to...

- ...avoid performance hazards through many indirections?
- ...translate architectural ideas into a real implementation?

Outline

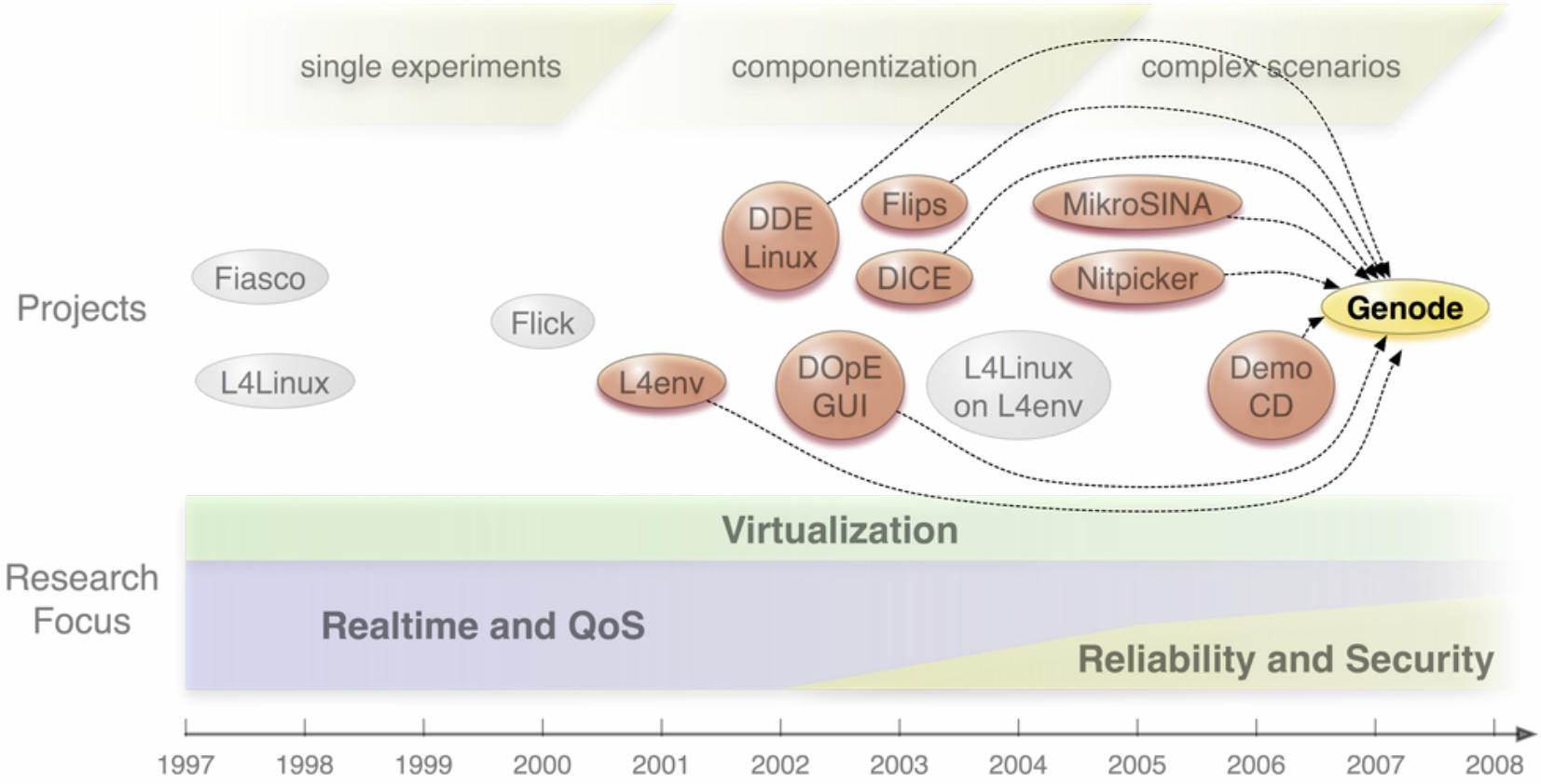
1. Why do we need another operating system?
2. Genode entering the picture
3. Architectural Principles
4. Core - the root of the process tree
5. Inter-process communication
6. Classification of components
7. Kernelization example
8. Components overview

A bit of history



Research timeline at TU Dresden

A new generation of kernels on the horizon



Unique feature: Cross-kernel portability

When started, no suitable microkernel was available

→ Prototyped on Linux and L4/Fiasco

→ Later ported to other kernels

Today: Rich OS construction kit

- Support of a variety of kernels

  OKL4, L4/Fiasco, L4ka::Pistachio, NOVA, Fiasco.OC, Linux, Codezero

- Preservation of special kernel features
  - OKLinux on OKL4,
  - L4Linux on Fiasco.OC,
  - Vancouver on NOVA,
  - Real-time priorities on L4/Fiasco

- Uniform API $\rightarrow$ kernel-independent components

- Many ready-to-use device drivers, protocol stacks, and 3rd-party libraries

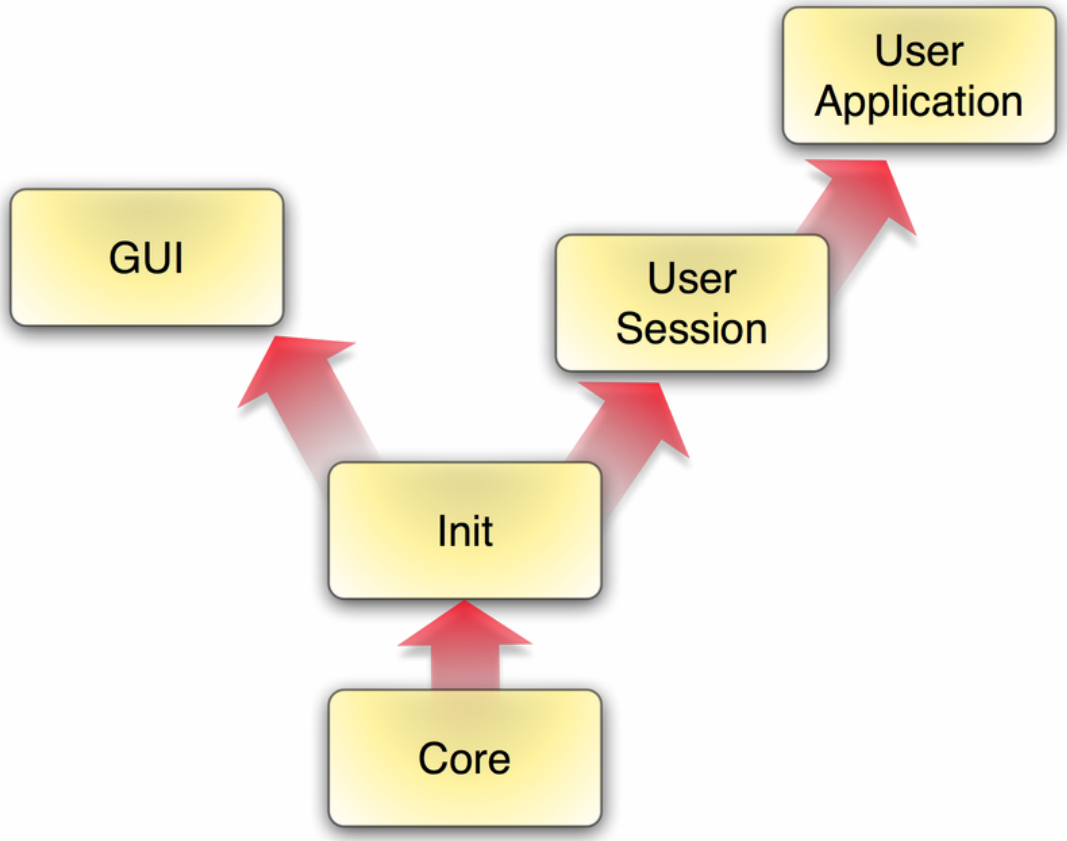
Outline

Object capabilities

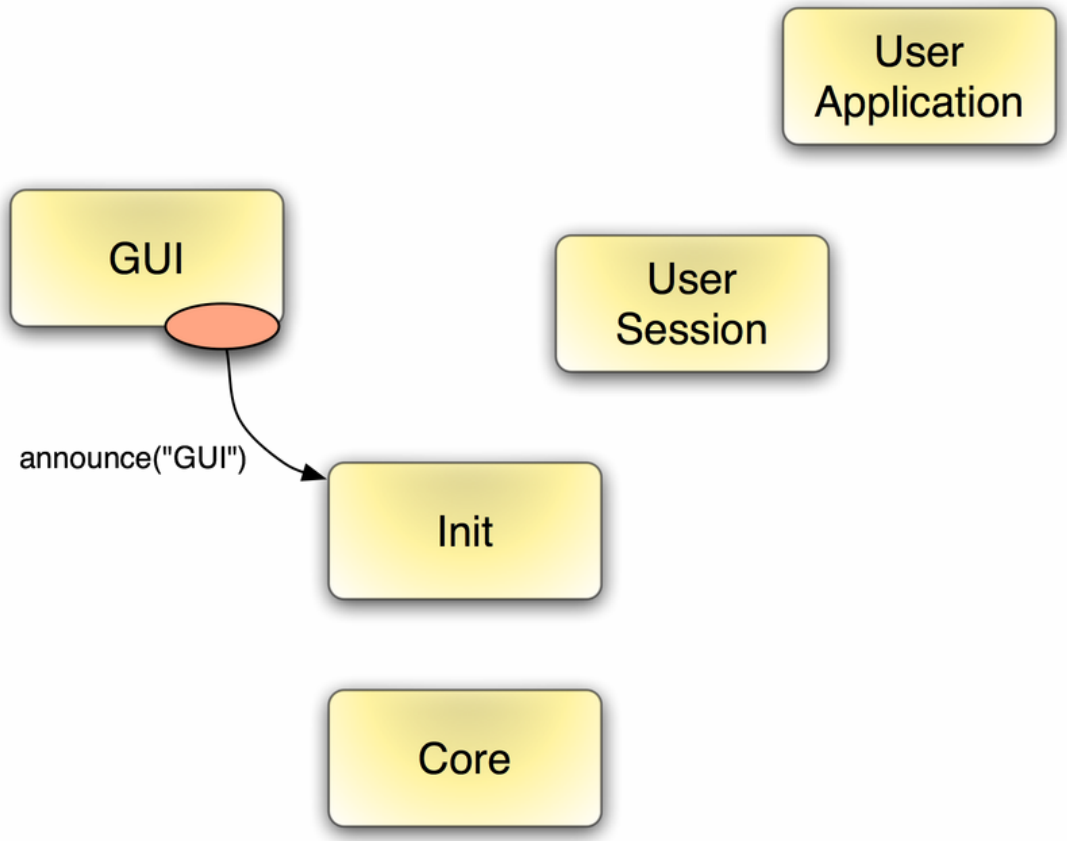
Delegation of rights

- Each process lives in a virtual environment
- A process that possesses a right *(capability)* can
  - Use it *(invoke)*
  - Delegate it to acquainted processes

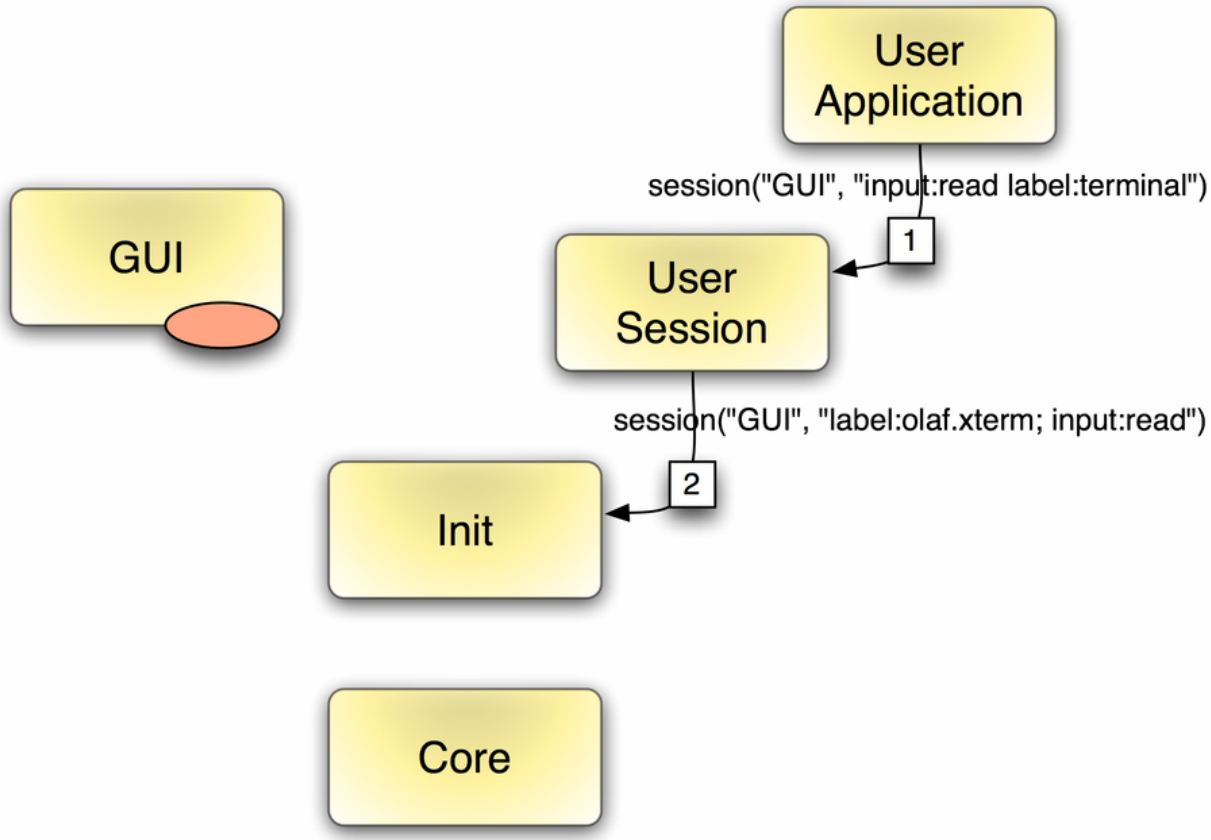
Recursive system structure



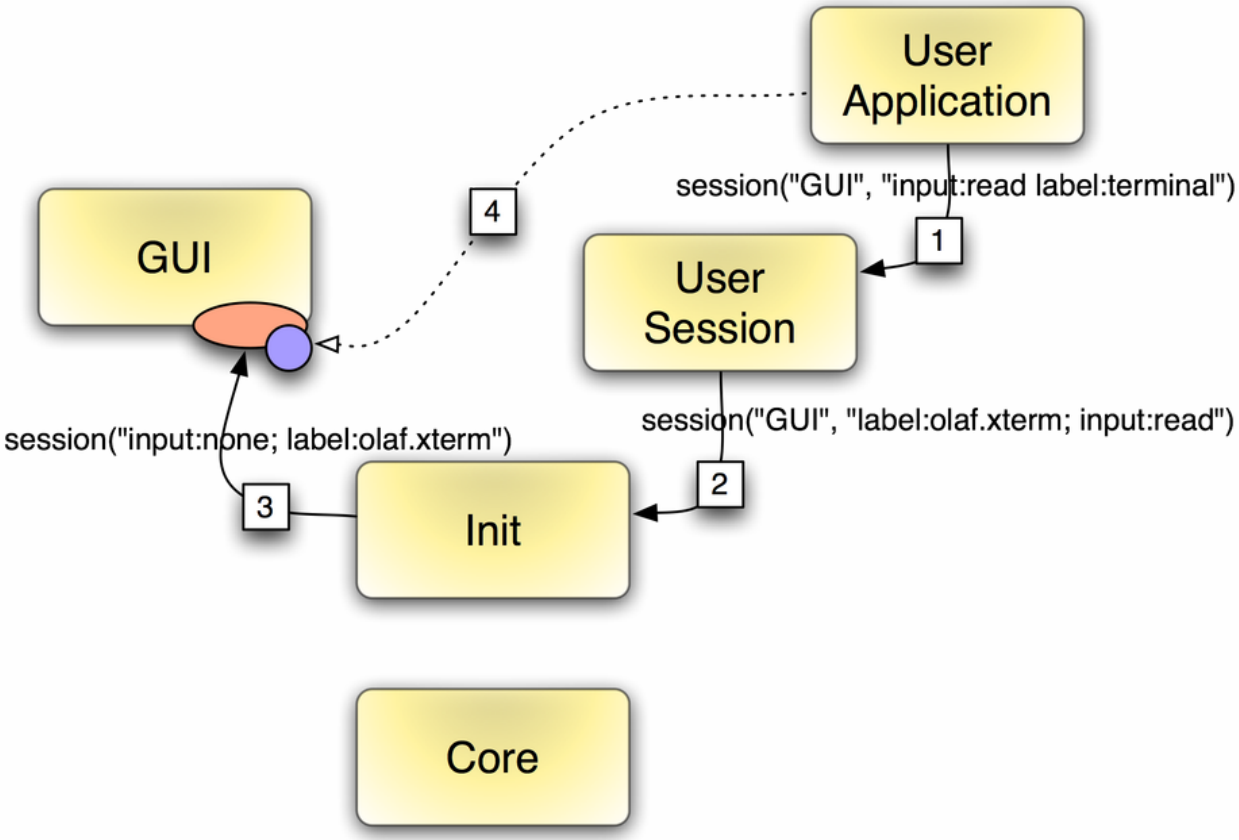
Service announcement



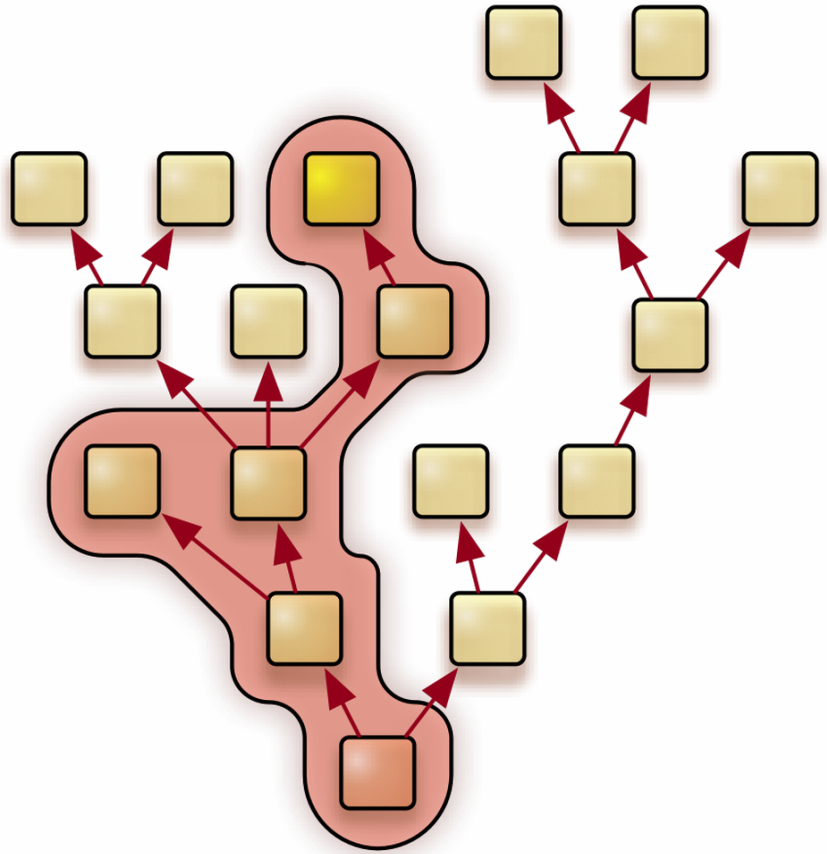
Session creation



Session creation

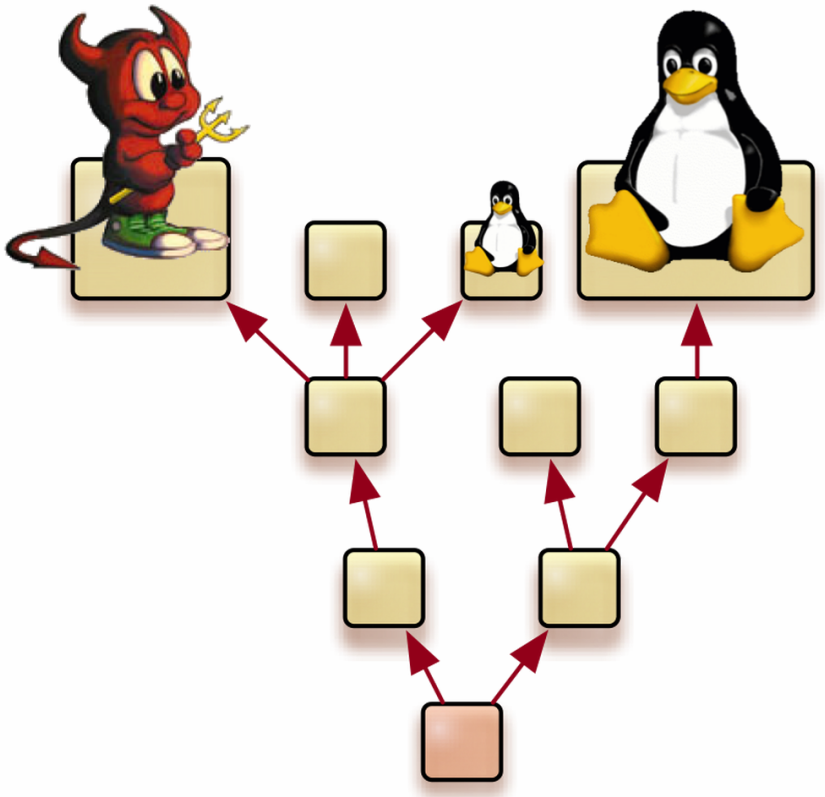


This works recursively



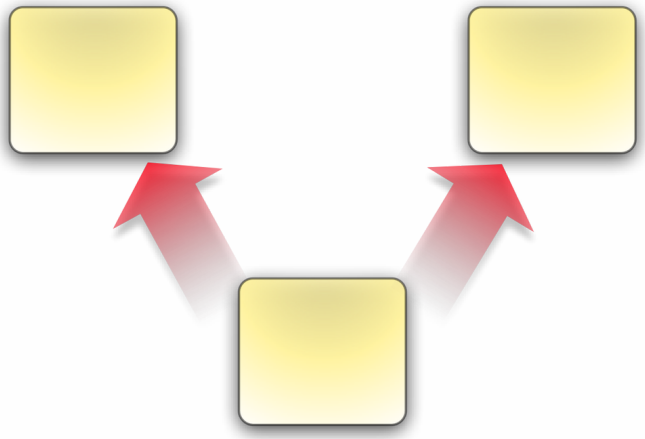
→ Application-specific TCB

Combined with virtualization



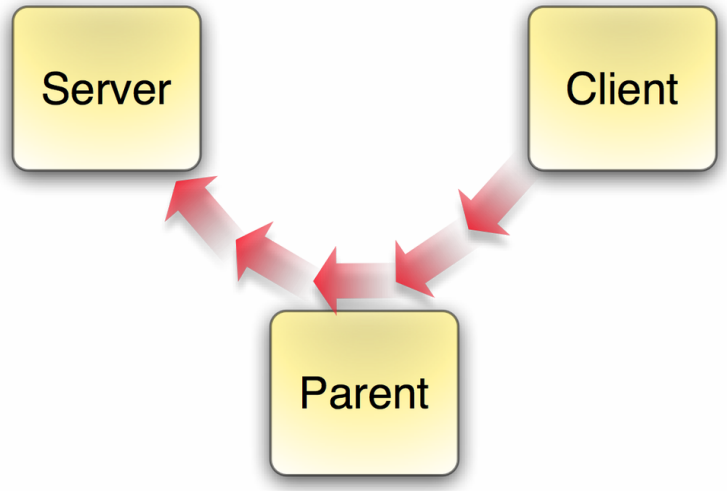
Resource management

Explicit assignment of physical resources to processes



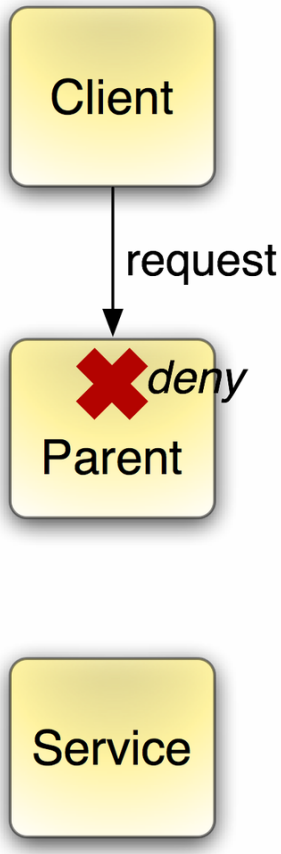
Resource management (II)

Resources can be attached to sessions



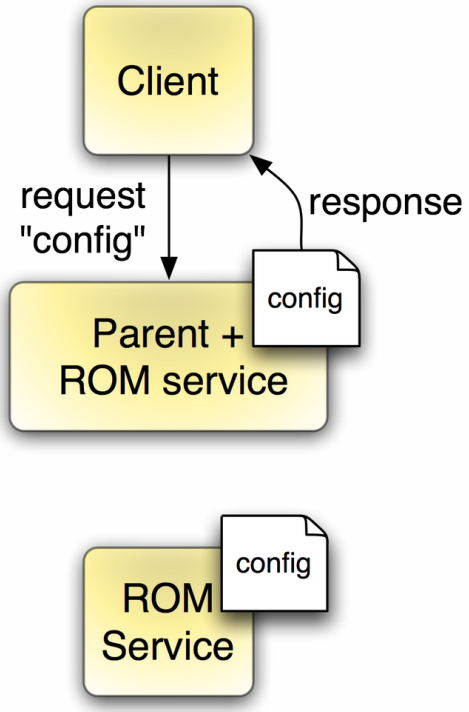
Resource management (III)

Intermediation of resource requests



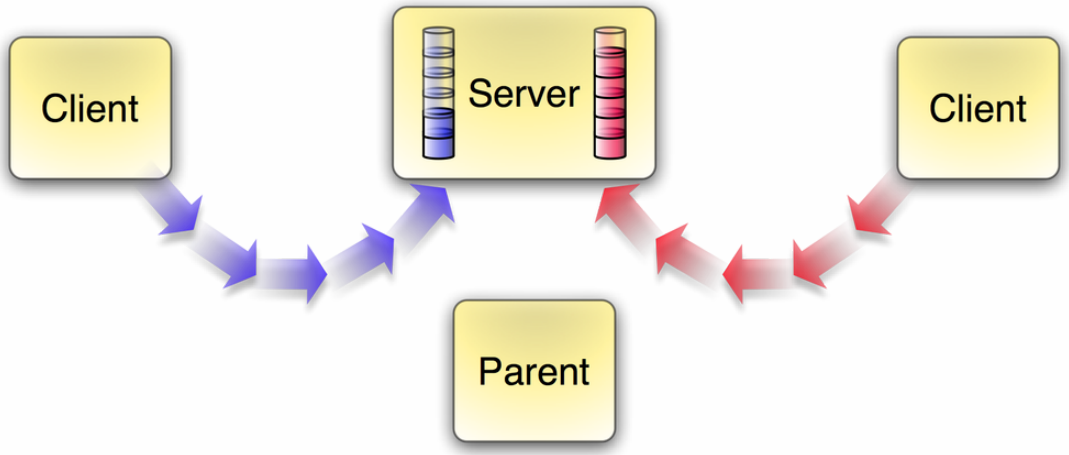
Resource management (IV)

Virtualization of resources

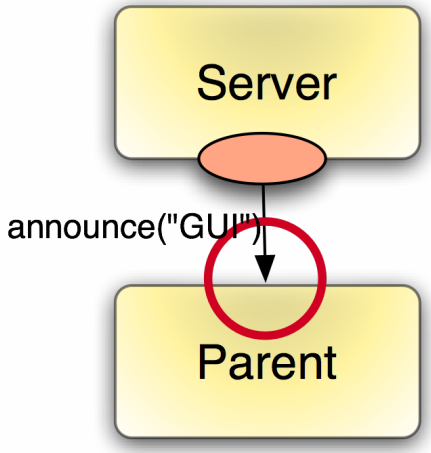


Resource management (V)

Server-side heap partitioning

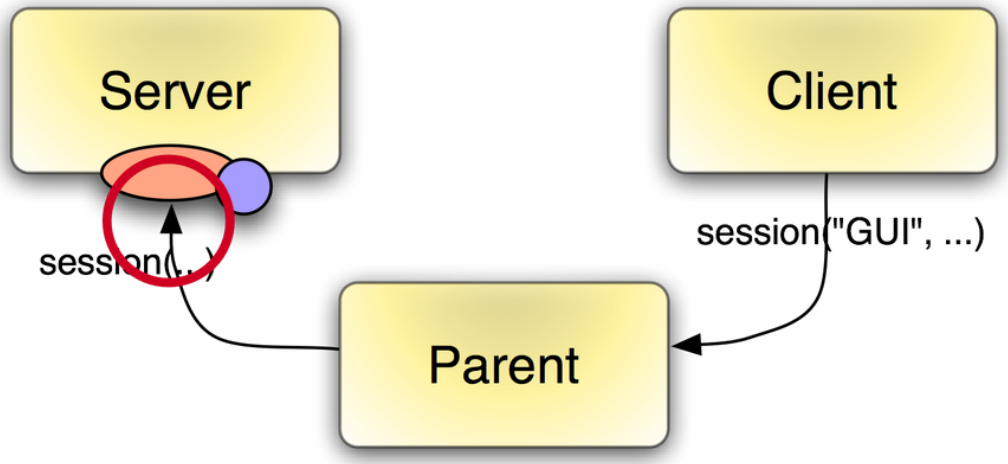


Parent interface




```
void exit(exit_value)

void announce(service_name, root_capability)

session_capability session(service_name, session_args)

void upgrade(to_session_capability, quantum)

void close(session_capability)
```

Root interface




```
session_capability session(session_args)

void upgrade(session_capability, upgrade_args)

void close(session_capability)
```

Outline

Core services

LOG RAM CAP CPU IO_MEM IO_PORT IRQ PD ROM RM SIGNAL

Core services

LOG RAM CAP CPU IO_MEM IO_PORT IRQ PD ROM RM SIGNAL

Debug output

```
amount write(string)
```

Core services

LOG **RAM** CAP CPU IO_MEM IO_PORT IRQ PD ROM RM SIGNAL

Physical memory

```
ram_dataspace_capability alloc(size, cached)

void free(ram_dataspace_capability)

void ref_account(ram_session_capability)

void transfer_quota(ram_session_capability, amount)

amount quota()

amount used()
```

Core services

LOG RAM **CAP** CPU IO_MEM IO_PORT IRQ PD ROM RM SIGNAL

Object identities

```
capability alloc(entrypoint_capability)

void free(capability)
```

Core services

LOG RAM CAP **CPU** IO_MEM IO_PORT IRQ PD ROM RM SIGNAL

Threads

```
thread_capability create_thread(name)

void kill_thread(thread_capability)

void start(thread_capability, ip, sp)
```

Core services

LOG RAM CAP CPU **IO_MEM** IO_PORT IRQ PD ROM RM SIGNAL

Memory-mapped I/O

Session arguments `base`, `size`, `write-combined`

```
io_mem_dataspace_capability dataspace()
```

Core services

LOG RAM CAP CPU IO_MEM **IO_PORT** IRQ PD ROM RM SIGNAL

Port-based I/O

Session arguments `base`, `size`

```
value inb(address)
value inw(address)
value inl(address)

void outb(address, value)
void outw(address, value)
void outl(address, value)
```

Core services

LOG RAM CAP CPU IO_MEM IO_PORT **IRQ** PD ROM RM SIGNAL

Device interrupts

Session argument `irq number`

```
void wait_for_irq()
```

Core services

LOG RAM CAP CPU IO_MEM IO_PORT IRQ **PD** ROM RM SIGNAL

Protection domain

```
void bind_thread(thread_capability)

void assign_parent(parent_capability)
```

Core services

LOG RAM CAP CPU IO_MEM IO_PORT IRQ PD **ROM** RM SIGNAL

Access to boot modules

Session argument `filename`

```
rom_dataspace_capability dataspace()
```

Core services

LOG RAM CAP CPU IO_MEM IO_PORT IRQ PD ROM **RM** SIGNAL

Address-space management

```
local_addr attach(dataspace_capability, size, offset,
                  use_local_addr, local_addr,
                  executable)

void detach(local_addr)

void add_client(thread_capability thread)

/* managed dataspaces */
dataspace_capability dataspace()
void fault_handler(signal_context_capability)
state state()
```

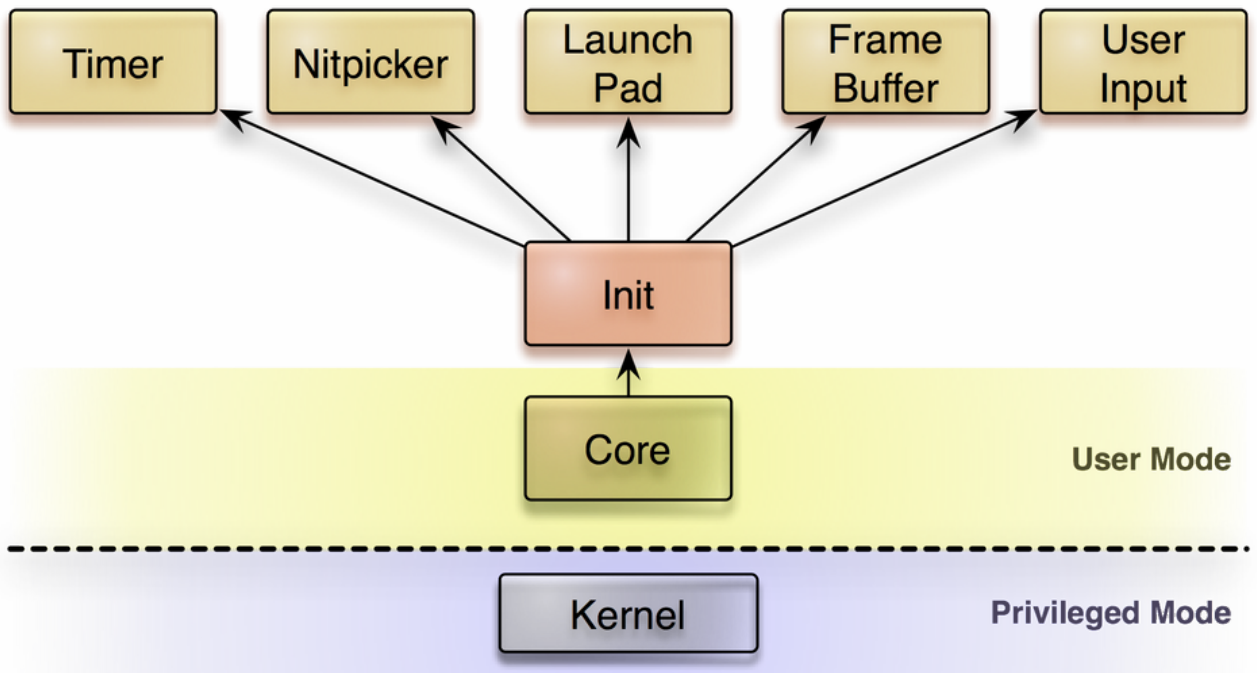
Core services

LOG RAM CAP CPU IO_MEM IO_PORT IRQ PD ROM RM **SIGNAL**

Asynchronous signal delivery

```
signal_context_capability alloc_context(imprint)

void free_context(signal_context_capability)

void submit(signal_context_capability, count)

signal wait_for_signal()
```

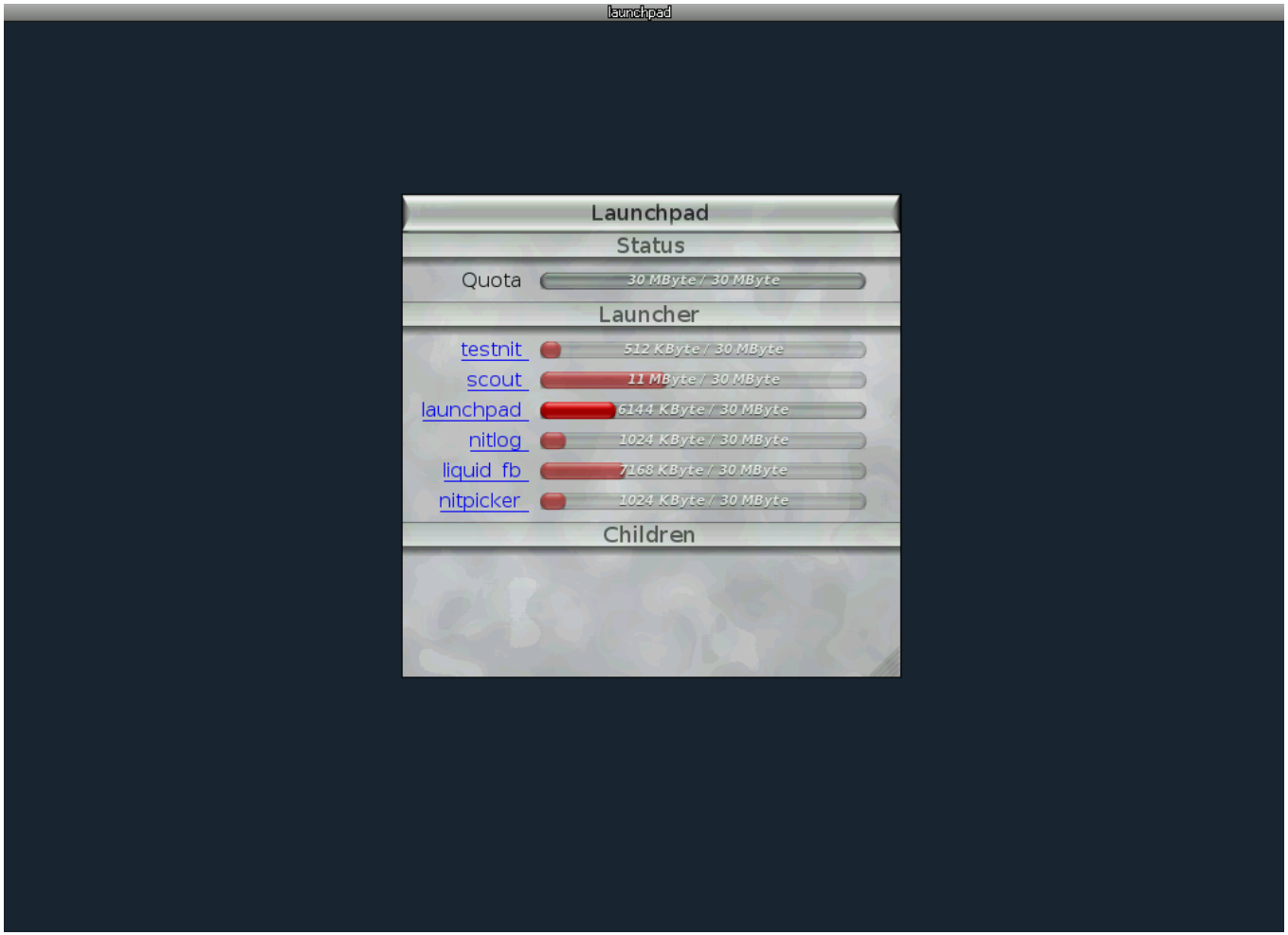
Default demo scenario



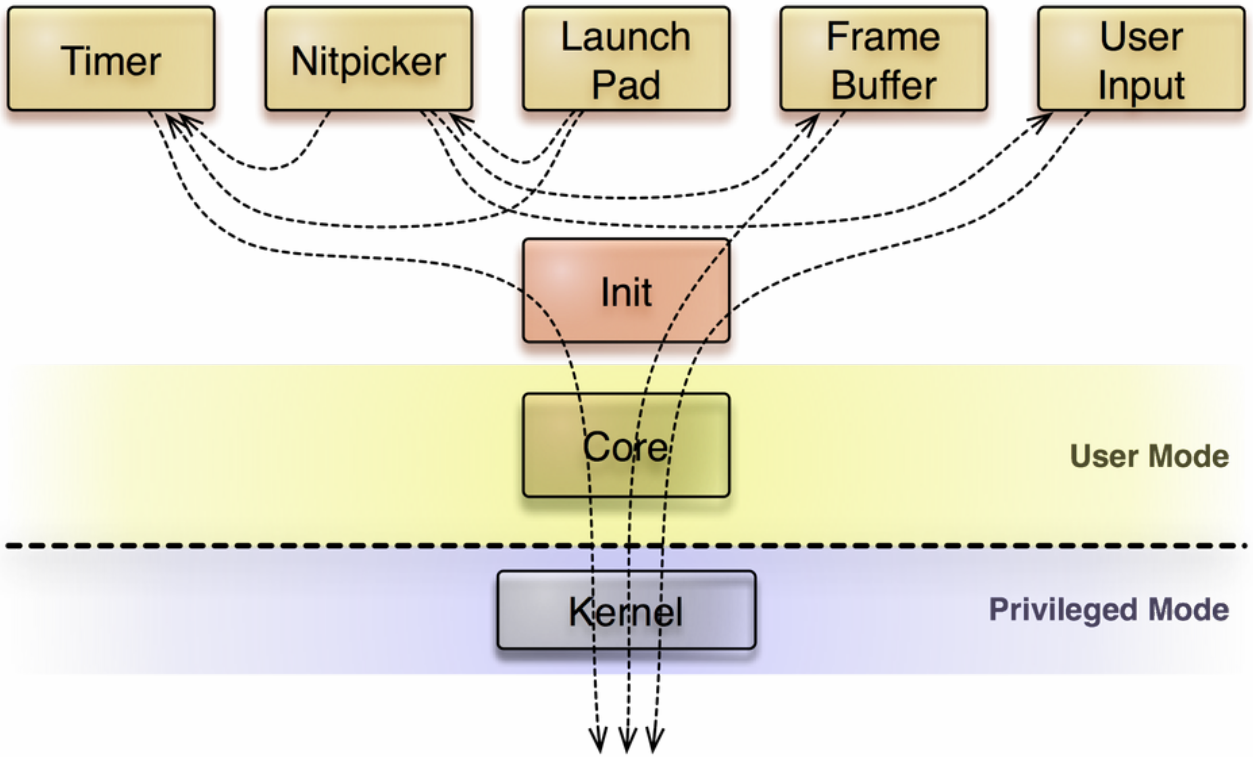
Configuration

```
<config>
  <parent-provides>
    <service name="ROM"/>     <service name="RAM"/>     <service name="IRQ"/>
    <service name="IO_MEM"/>  <service name="IO_PORT"/> <service name="CAP"/>
    <service name="PD"/>      <service name="RM"/>      <service name="CPU"/>
    <service name="LOG"/>
  </parent-provides>
  <default-route> <any-service> <parent/> <any-child/> </any-service> </default-route>
  <start name="pci_drv">
    <resource name="RAM" quantum="1M"/>
    <provides><service name="PCI"/></provides> </start>
  <start name="vesa_drv">
    <resource name="RAM" quantum="1M"/>
    <provides><service name="Framebuffer"/></provides> </start>
  <start name="ps2_drv">
    <resource name="RAM" quantum="1M"/>
    <provides><service name="Input"/></provides> </start>
  <start name="timer">
    <resource name="RAM" quantum="1M"/>
    <provides><service name="Timer"/></provides> </start>
  <start name="nitpicker">
    <resource name="RAM" quantum="1M"/>
    <provides><service name="Nitpicker"/></provides> </start>
  <start name="launchpad">
    <resource name="RAM" quantum="32M"/> </start>
</config>
```

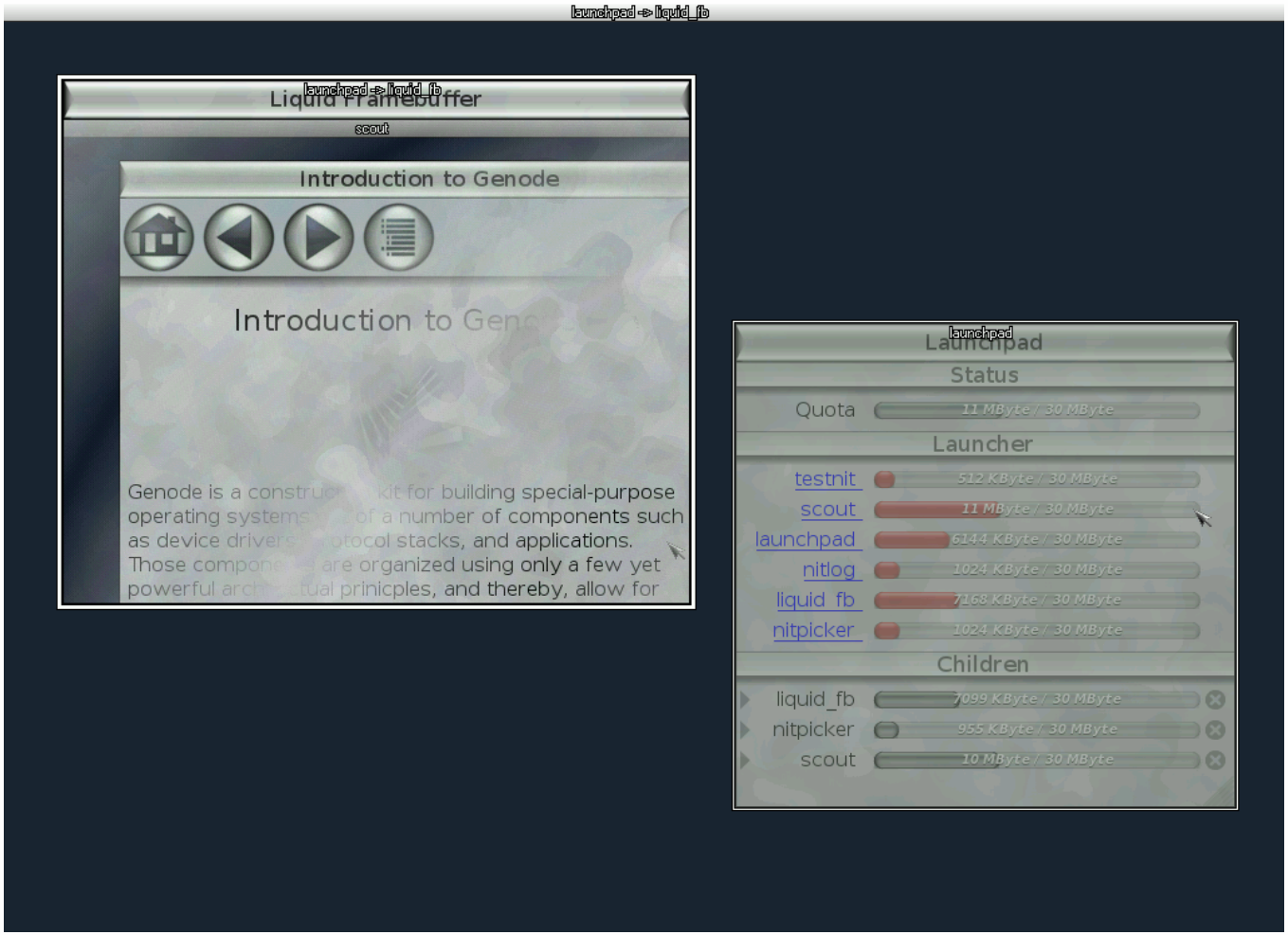
Screenshot


launchpad
Launchpad
Status
Quota
30 MByte / 30 MByte
Launcher
testnit
512 KByte / 30 MByte
scout
11 MByte / 30 MByte
launchpad
6144 KByte / 30 MByte
nitlog
1024 KByte / 30 MByte
liquid_fb
7168 KByte / 30 MByte
nitpicker
1024 KByte / 30 MByte
Children

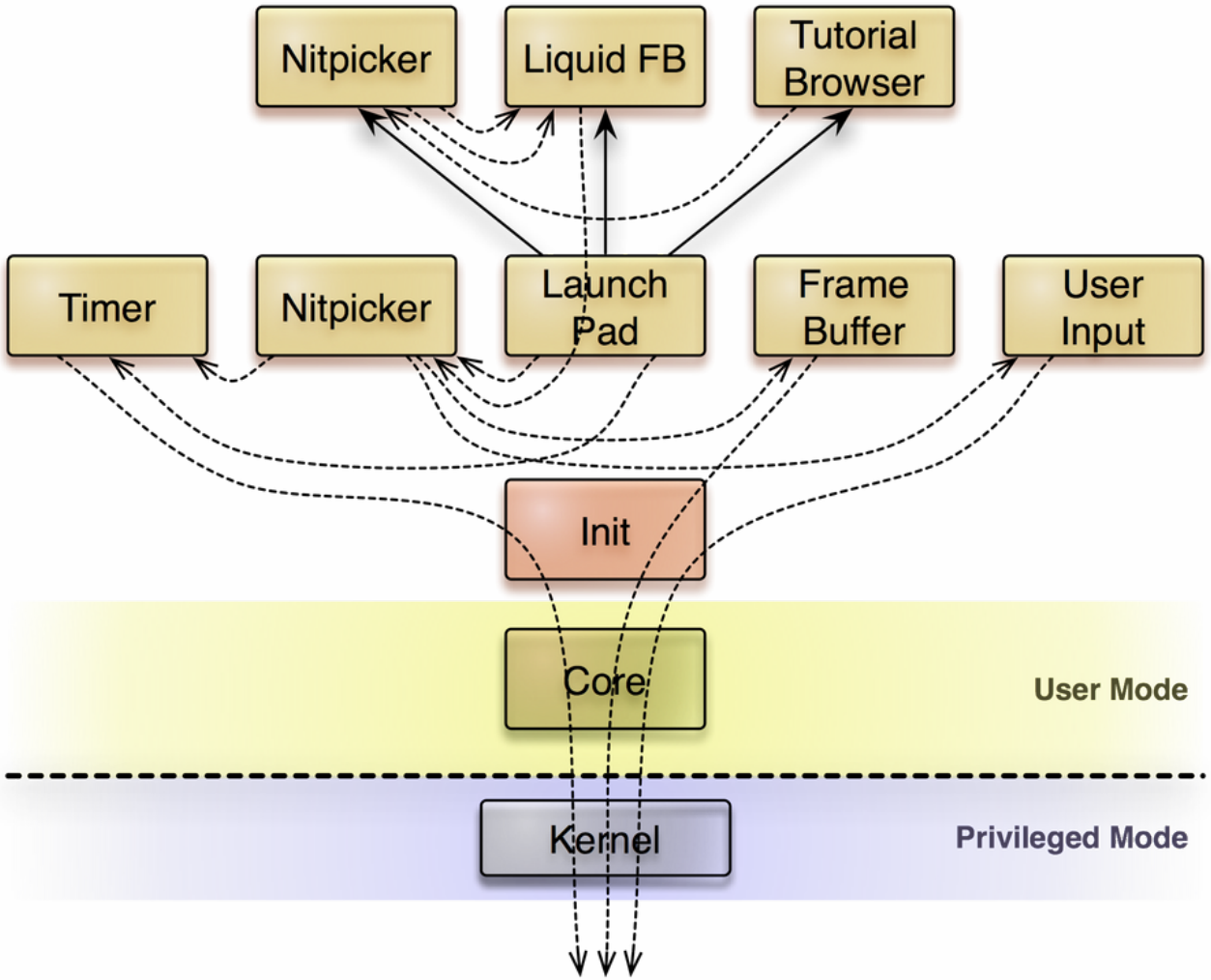
Sessions


Timer
Nitpicker
Launch Pad
Frame Buffer
User Input
Init
Core
User Mode
Kernel
Privileged Mode

Virtualized framebuffer

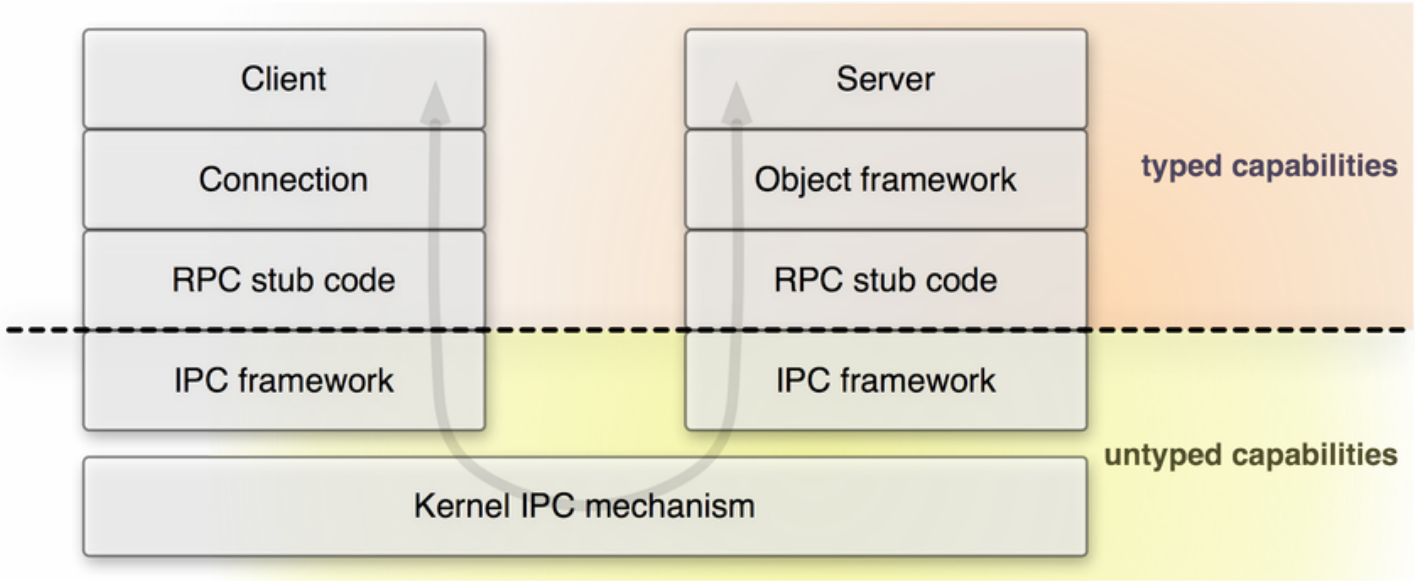


Sessions including virtualized framebuffer

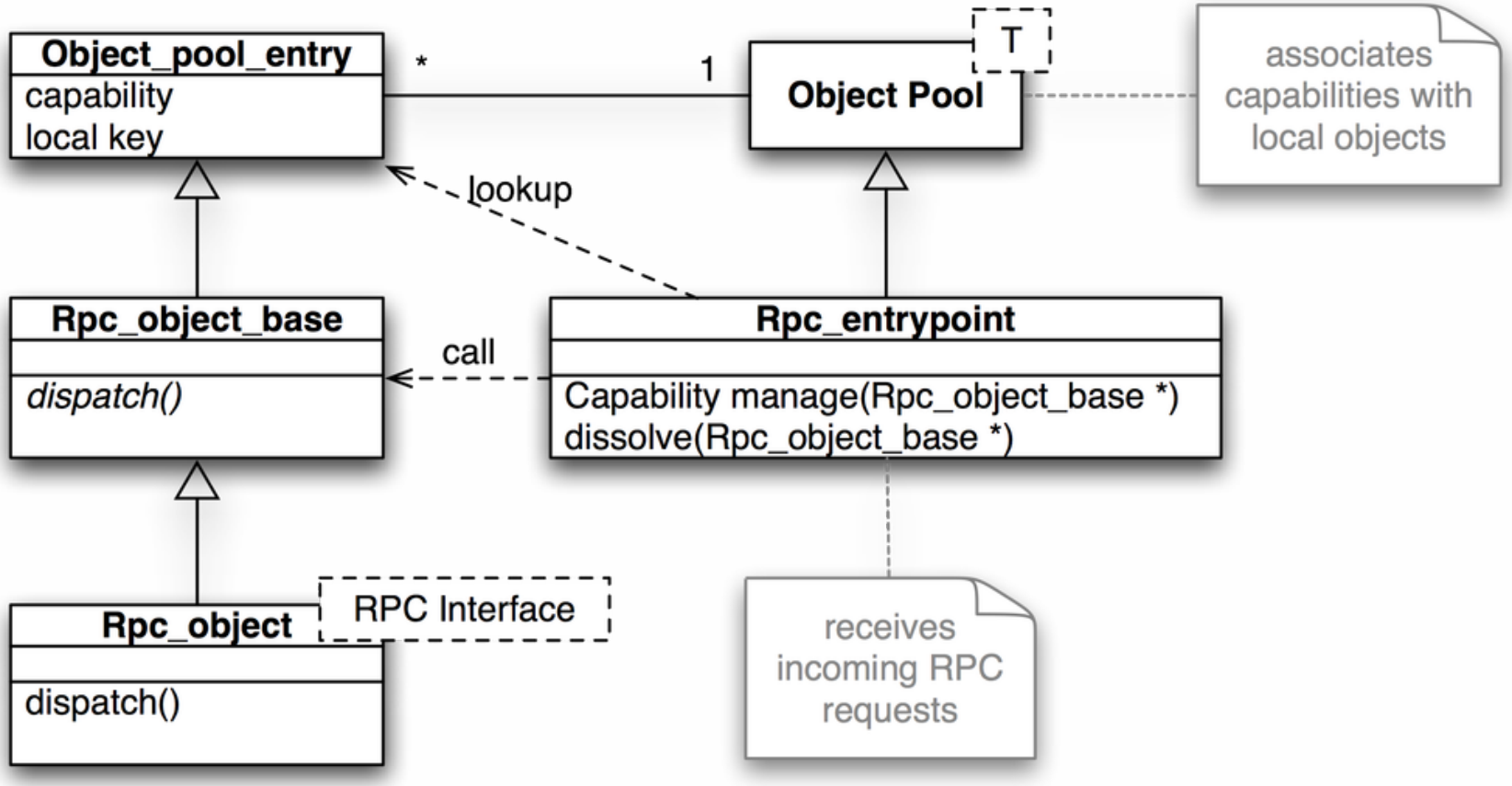


Outline

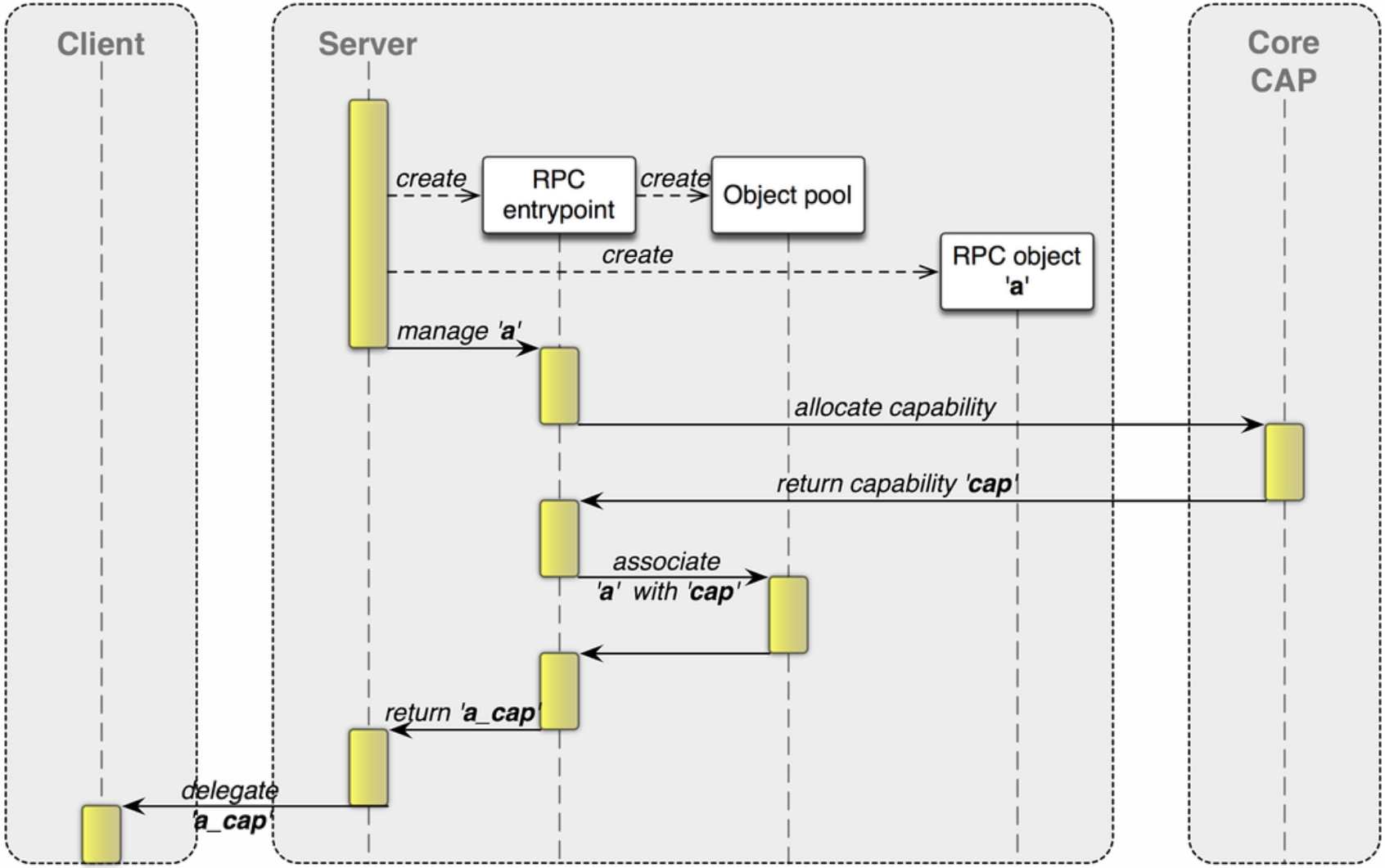
Remote procedure calls (RPC)



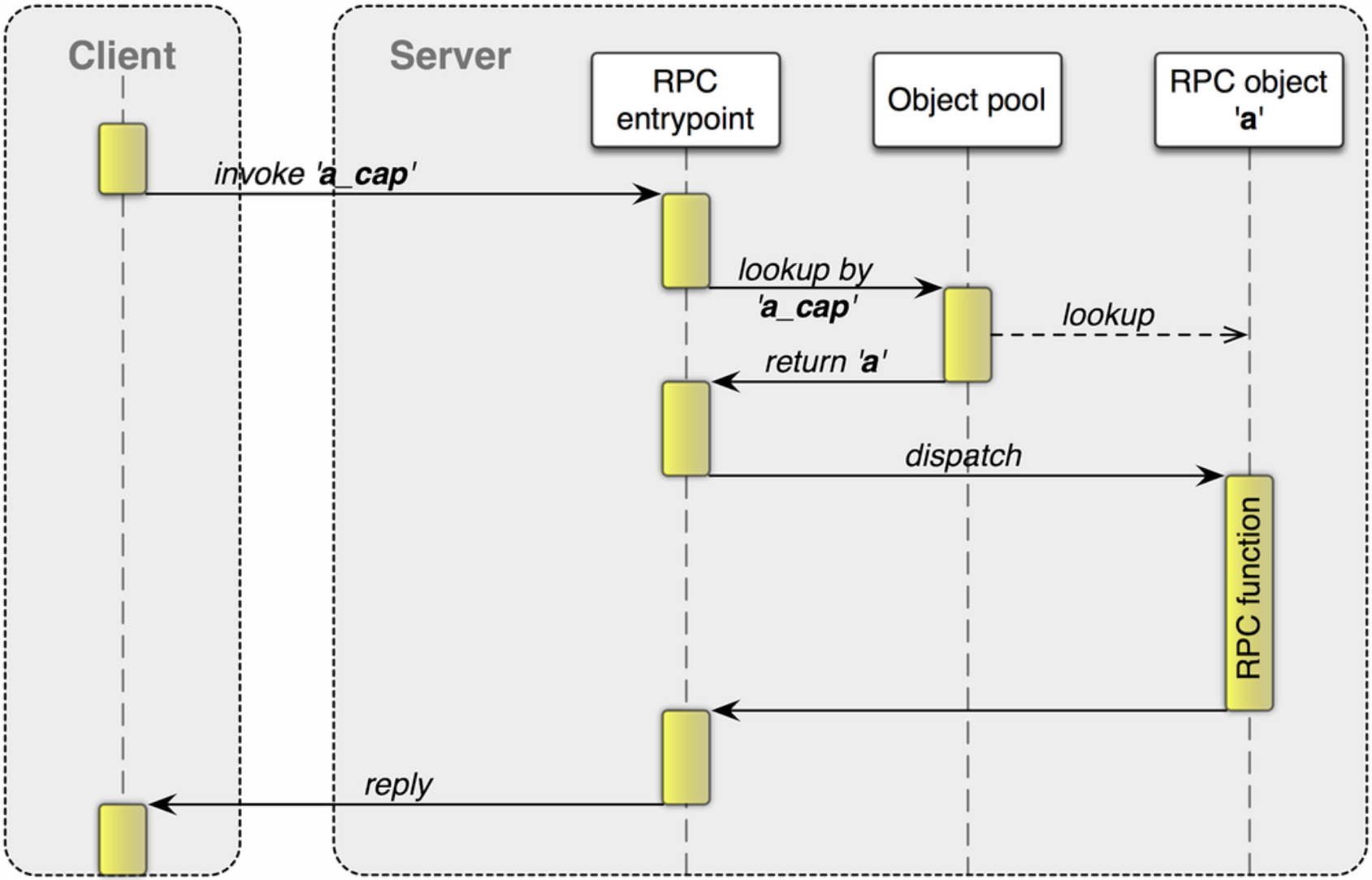
Remote procedure calls: Classes



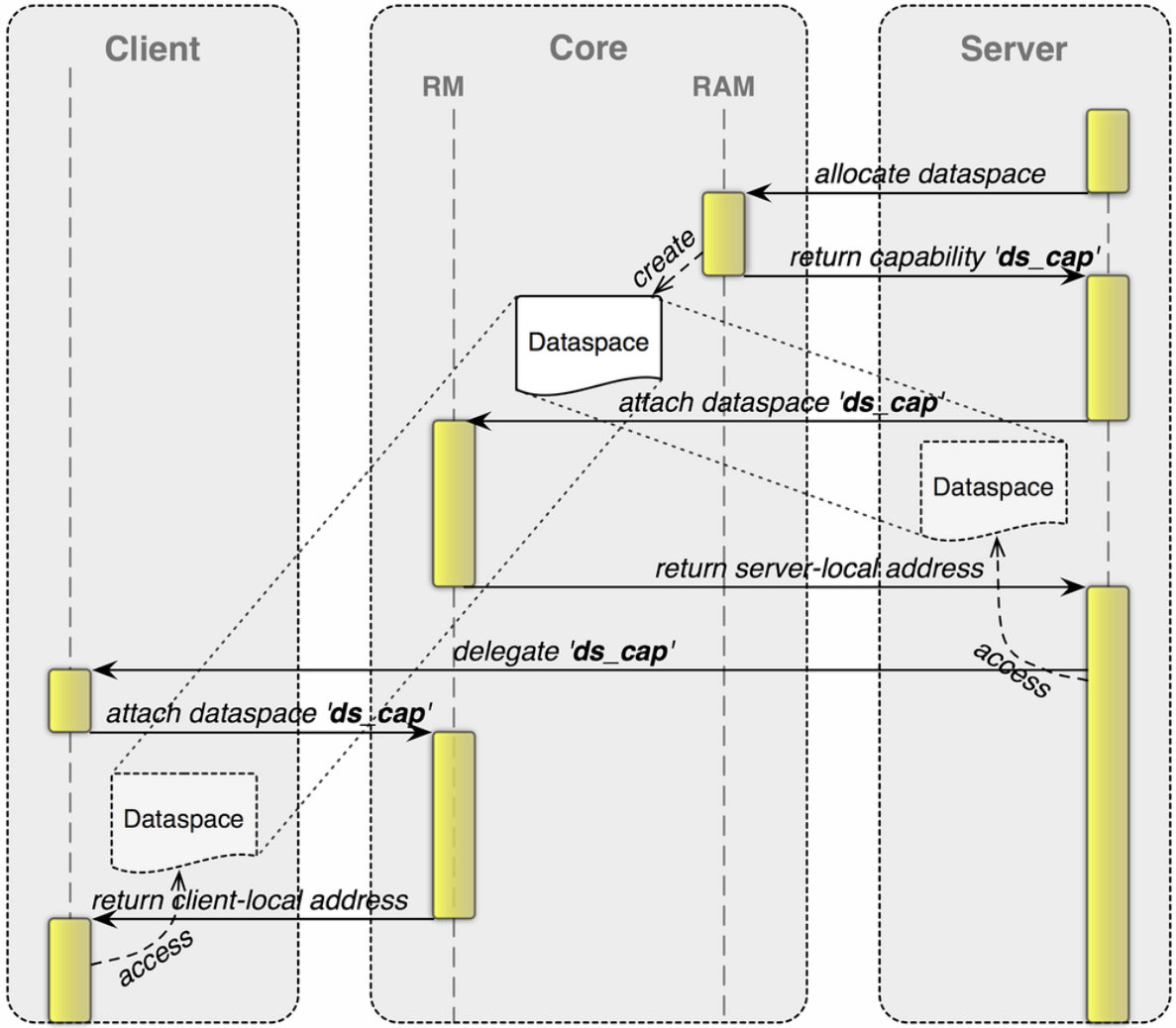
Remote procedure calls: New RPC object



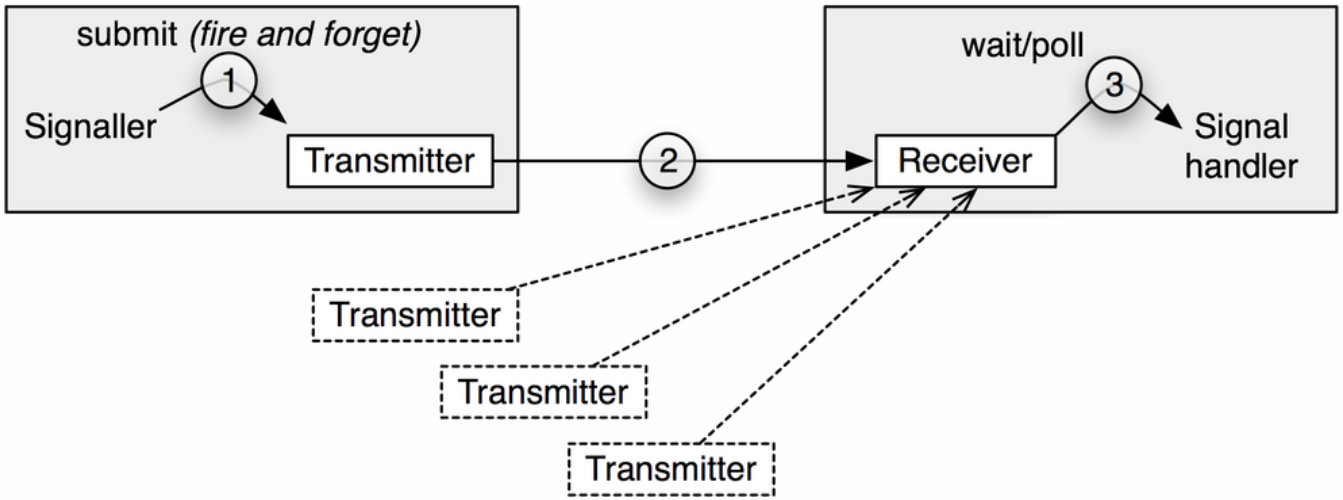
Remote procedure calls: Invocation



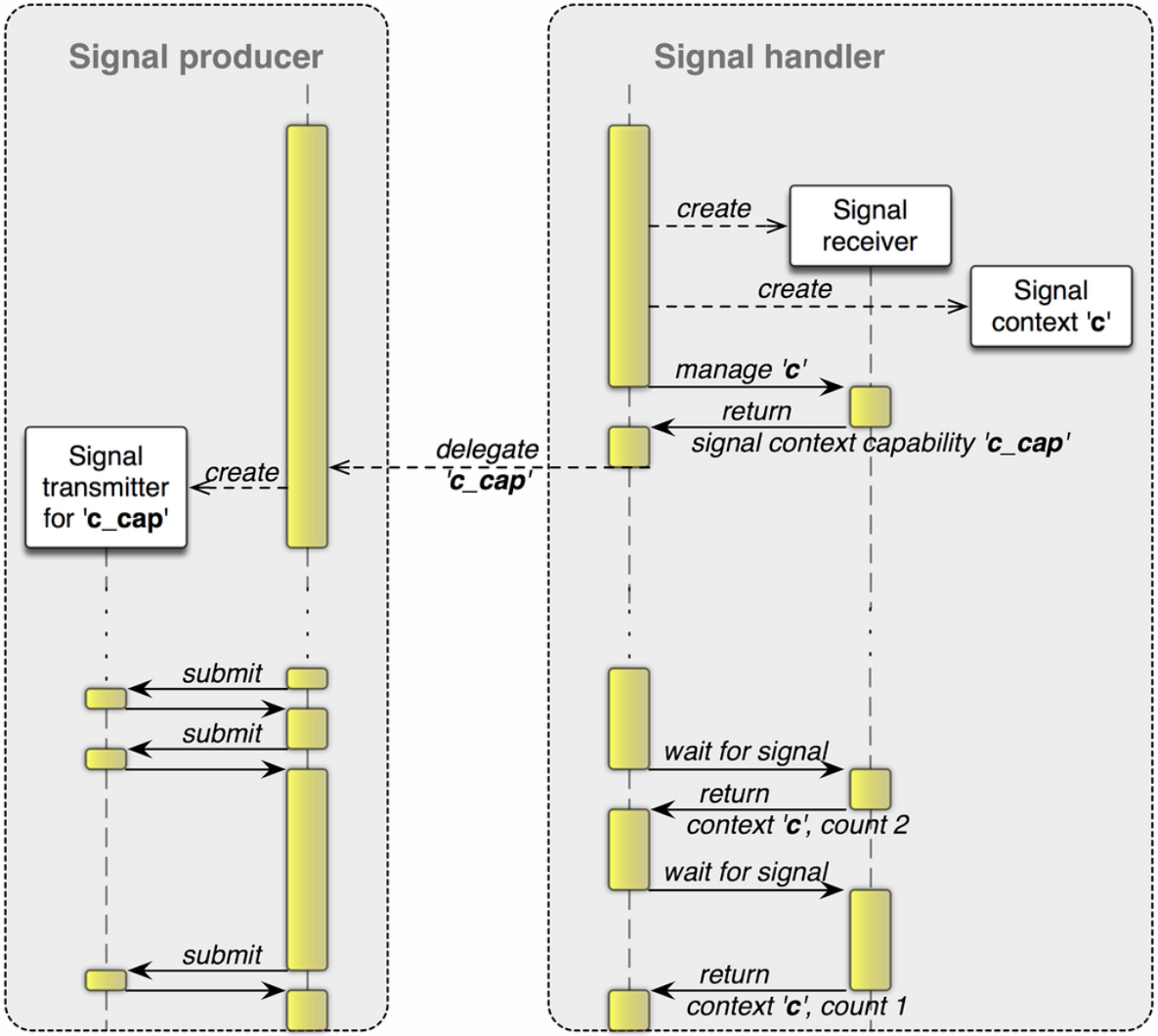
Shared memory



Asynchronous notifications



Asynchronous notifications (II)

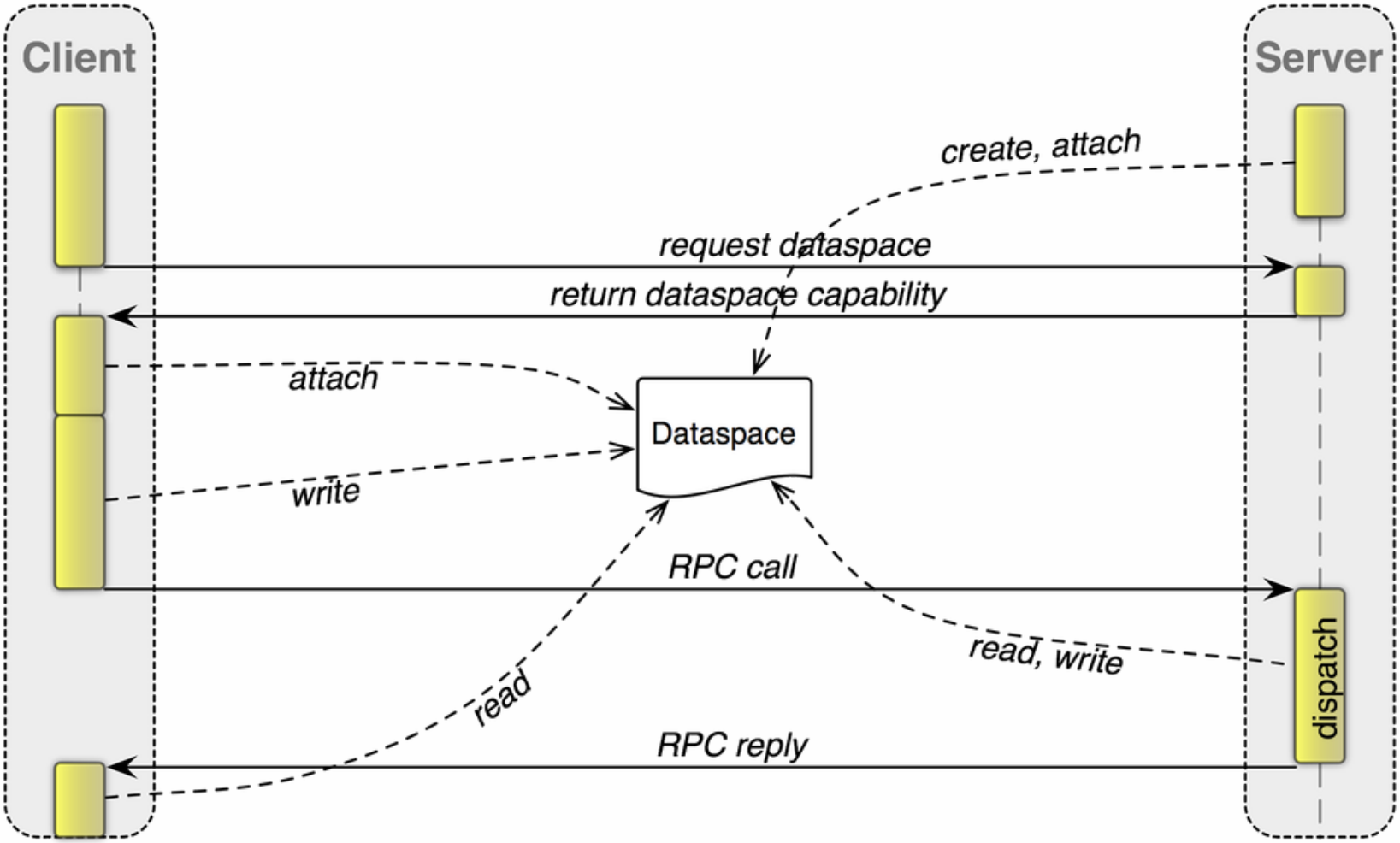


Mechanisms combined

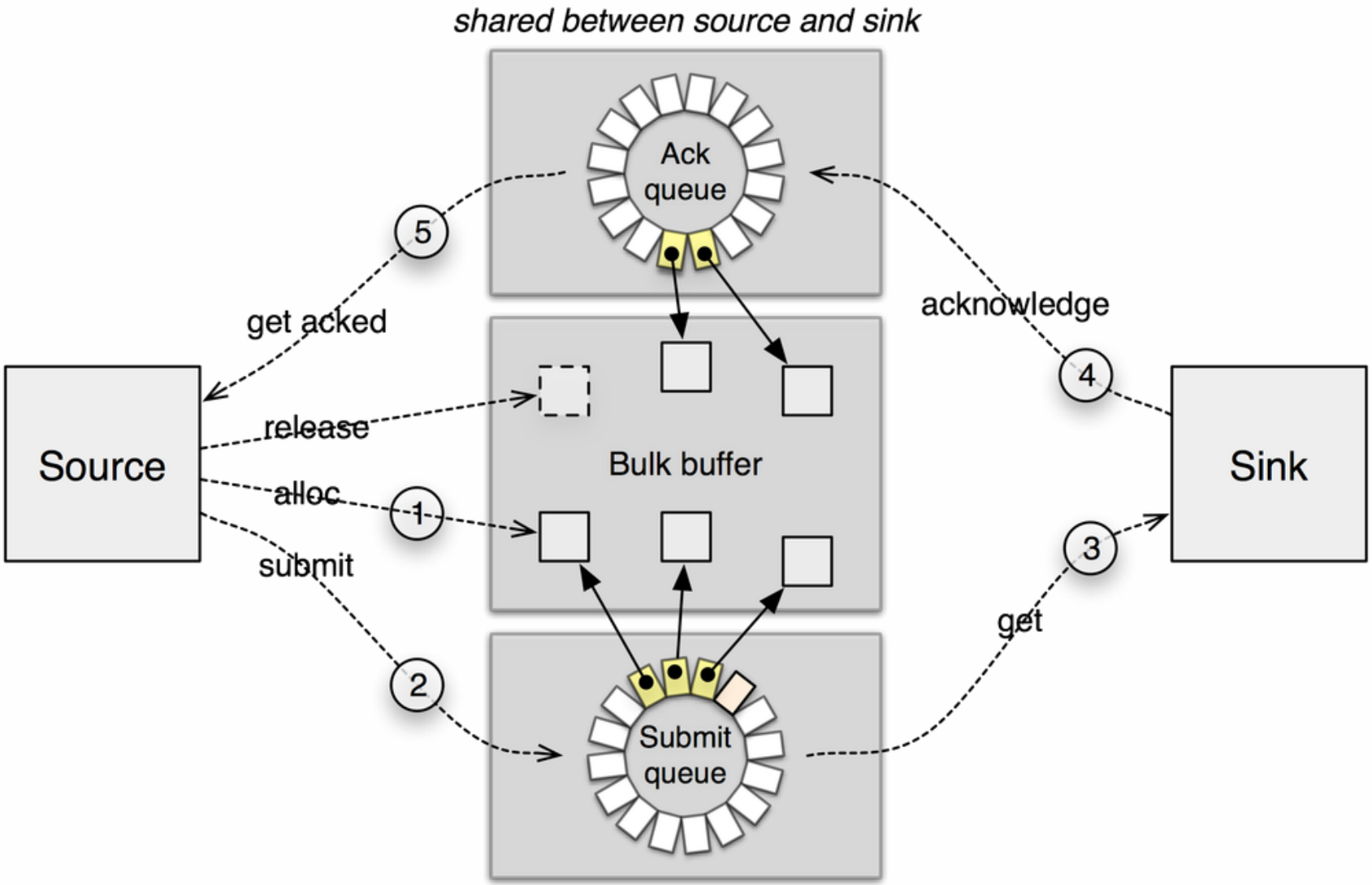
- RPC + shared memory
  → Synchronous bulk data (transaction)

- Asynchronous notifications + shared memory
  → Asynchronous bulk data (streaming)

Synchronous bulk data transfer



Asynchronous bulk data transfer



Packet stream in detail

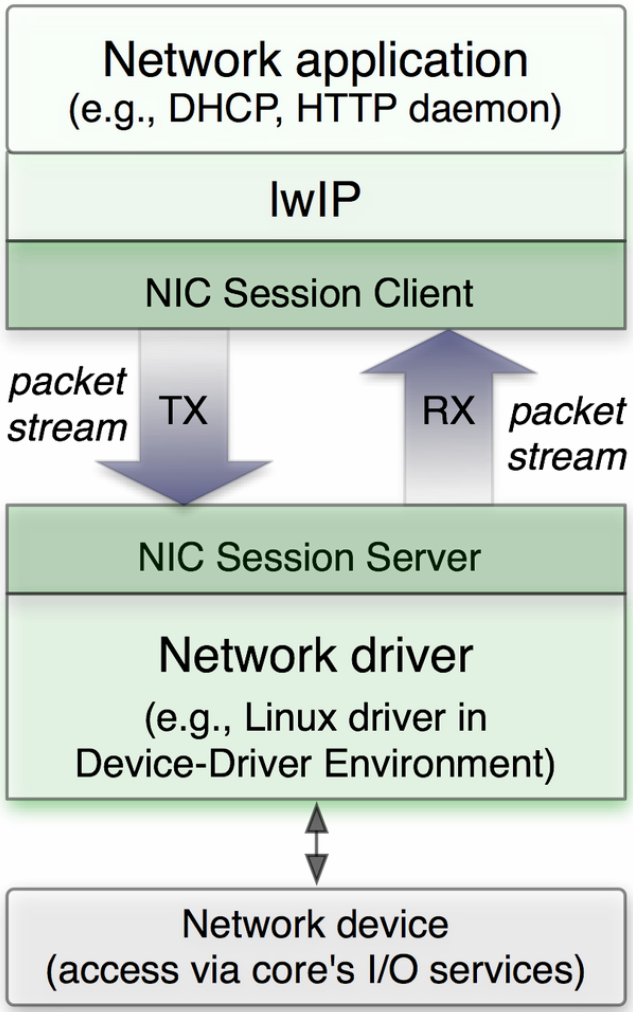
Packet descriptor

- Allocated by source
- Enqueued in submit / acknowledgement queue
- Describes portion of bulk buffer (offset, size)
- Carries domain-specific control information

Conditions

- Submit queue is full
- Submit queue is empty
- Acknowledgement queue is full
- Acknowledgement queue is empty
  $\rightarrow$ wakeup via signals

Packet stream example



Outline

Classification

Kernel enables base platform

Device driver translates device interface to API

Protocol stack translates API to API

Application is leaf node in process tree

Runtime environment has one or more children

Resource multiplexer has multiple clients

combinations are possible

Kernel

FIASCO.OC

OKL4

CODEZERO

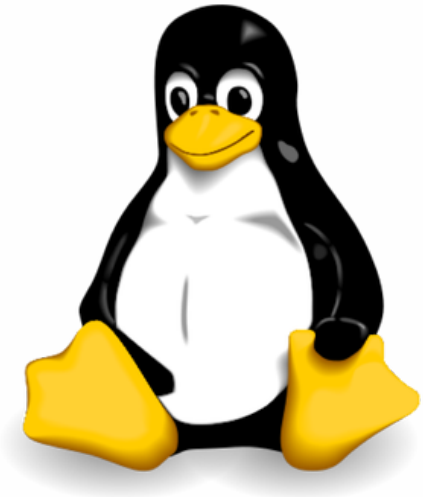
FIASCO

NOVA

Microhypervisor

L4Ka

MicroBlaze



Device driver

Translates device interface to session interface

- Uses core's IO_MEM, IO_PORT, IRQ services
- Single client
- Contains no policy
- Enforces policy (device-access arbitration)

Device driver (2)

Critical because of DMA

- MMU protects physical memory from driver code
- Driver code accesses device via MMIO
- Device has access to whole physical memory (DMA)

→ Device driver can access whole physical memory

IOMMUs can help ...but are no golden bullet

Device driver (3)

Even with no IOMMU, isolating drivers has benefits

- Taming classes of non-DMA-related bugs
  - Memory leaks
  - Synchronization problems, dead-locks
  - Flawed driver logic, wrong state machines
  - Device initialization
- Minimizing attack surface from the outside

Protocol stack

Translates API to another (or the same) API

- Does not enforce policy
- Single client
- May be co-located with device driver

Protocol stack (2)

Libraries

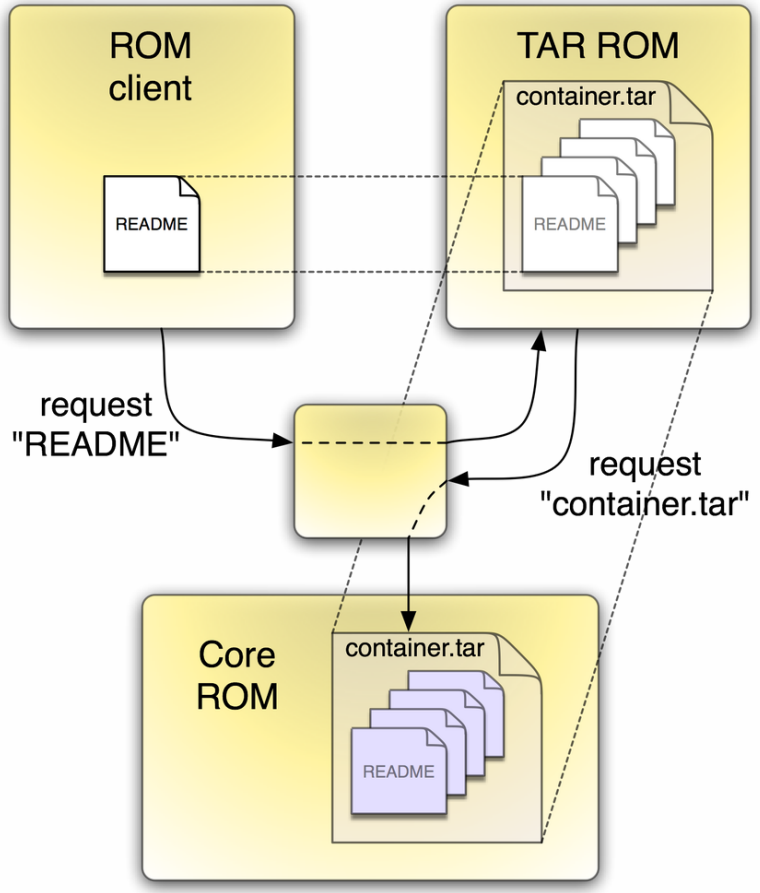
Library	Translation
Qt4	Qt4 API → various Genode sessions
lwIP	socket API → NIC session

Components translating sessions

Component	Translation
TCP terminal	Terminal session → NIC session
iso9660	ROM session → Block session
ffat_fs	File-system session → Block session

Protocol stack (3)

Components that filter sessions



Protocol stack (4)

Operate on session interfaces, not physical resources

→ May be instantiated any number of times

→ Critical for availablility

→ Not neccessarily critical for integrity and confidentiality

→ Information leakage constrained to used interfaces

complex code should go in here

Application

Leaf node in process tree

- Uses services
- Implements application logic
- Provides no service

Runtime environment

Hosts other processes as children

Defines and imposes policy!

Examples

- Init
- Virtual machine monitor
- Debugger
- Python interpreter

Resource multiplexer

Multiplexes session interface

- Multiple clients $\rightarrow$ Potential multi-level component
- Free from policy
- Enforce policy dictated by parent
- Prone to cross-client information leakage
- Prone to resource-exhaustion-based DoS

Resource multiplexer (2)

→ Often as critical as the kernel

→ Must be as low complex as possible

→ Must work on client-provided resources

→ Must employ heap partitioning

only a few resource multiplexers needed

Outline

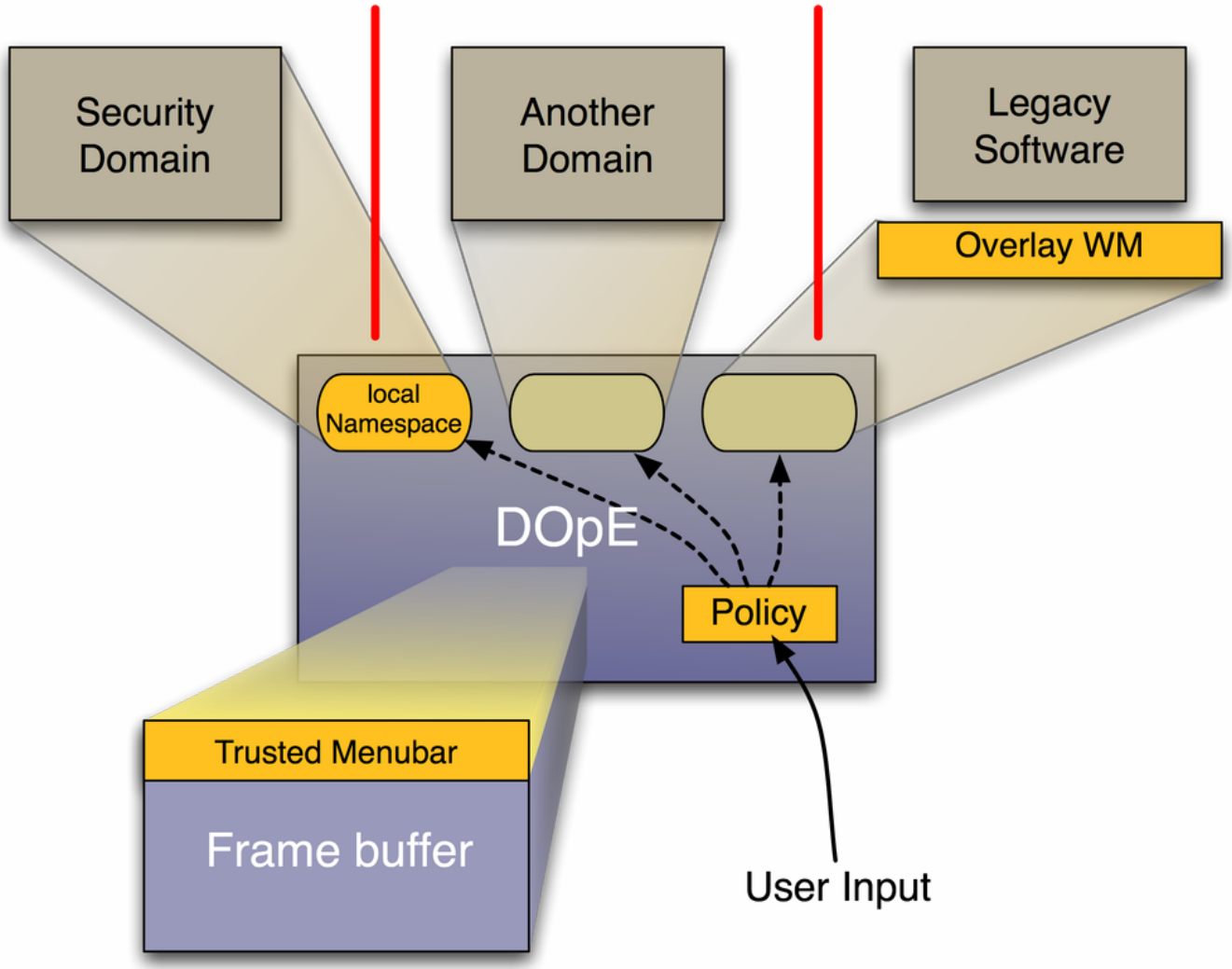
Case study: Kernelizing the GUI server

Persistent security problems of GUIs

- Impersonation
  (Trojan horses, phishing, man in the middle)
- Spyware
  (input loggers, arcane observers)
- Robustness/availability risks
  (resource-exhaustion-based denial of service

GUI belongs to TCB $\rightarrow$ *low complexity is important!*

Starting point: DOpE as secure GUI



DOpE as secure GUI - Drawbacks

- Prone to resource exhaustion by malicious clients
- Provides custom look and feel*
  - Stands in the way when using legacy software
  - May be enhanced by theme support
- Complexity of 12,000 LOC

Straight-forward attempt: Shrinking DOpE

Revisiting the implementation

- Keeping only essential functionality
  $\rightarrow$ 7,000 LOC

We loose:

- Majority of widgets (grid, scale, scrollbar, etc.)
- Flexible command interface
- Coolness, fancyness, convenience
- Real-time support

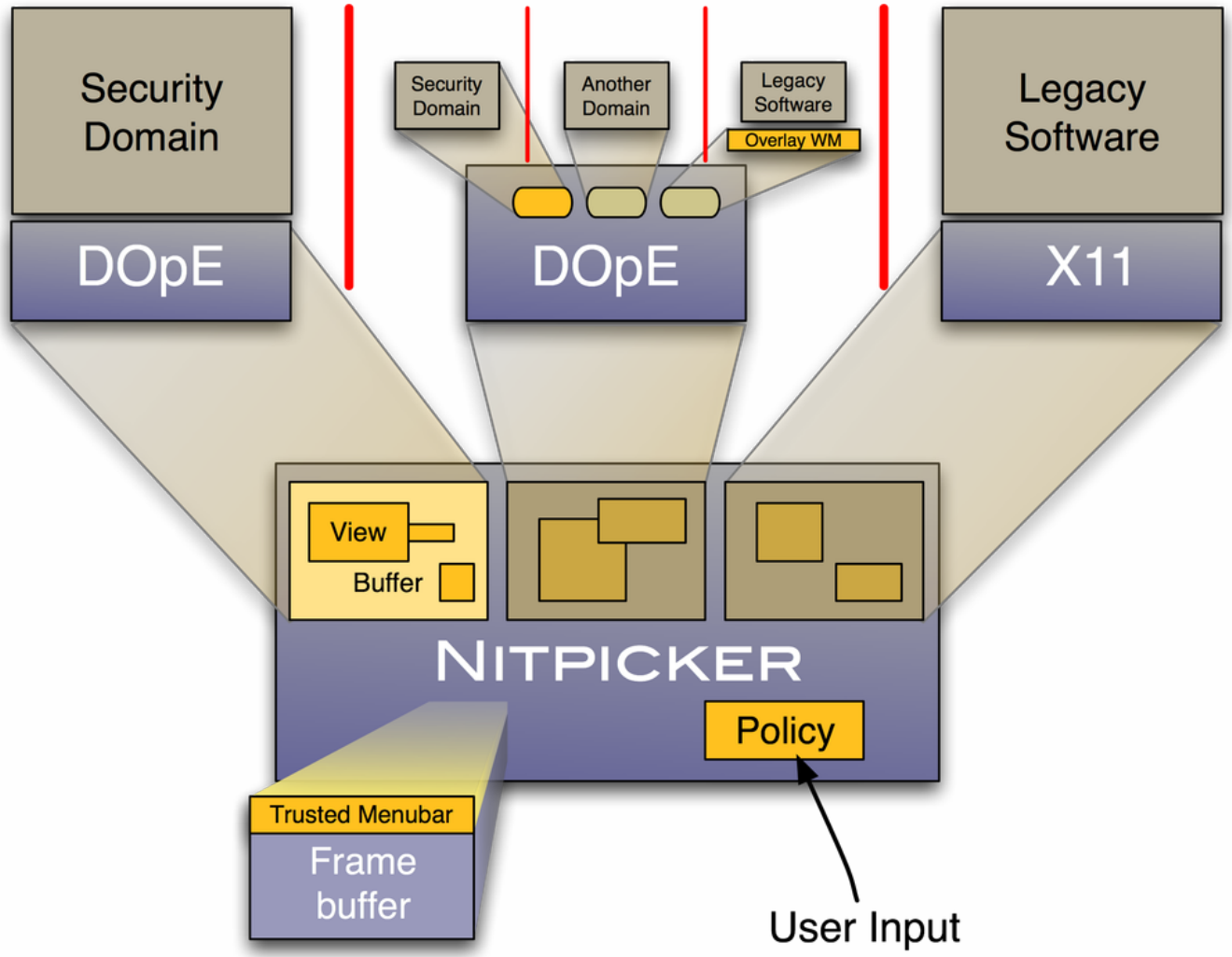
7,000 LOC are too much for such a crippled GUI!

Bottom-up approach

What do we really need in the GUI server?

- Widgets? → No
- Font support? → No
- Window decoration? → No
- Textual command interface? → No
- Look and feel, gradients, translucency? → No
- Hardware abstractions (e. g., color-space conversion)? → No
- Windows displaying pixel buffers? → YES
- Distribution of input events? → YES
- Secure labeling? → YES

Buffers and views



User interaction

Input-event handling

- Only **one receiver** of each input event
- **Focused view** defines input routing
- Routing controlled **by the user** only

Client-side window handling

Report motion events to focused view while a button is pressed
→ Client-side window policies (move, resize, stacking)
→ Key for achieving low server-side complexity

Emergency break
→ Special key regains control over misbehaving applications

Trusted path

It is not sufficient to label windows!

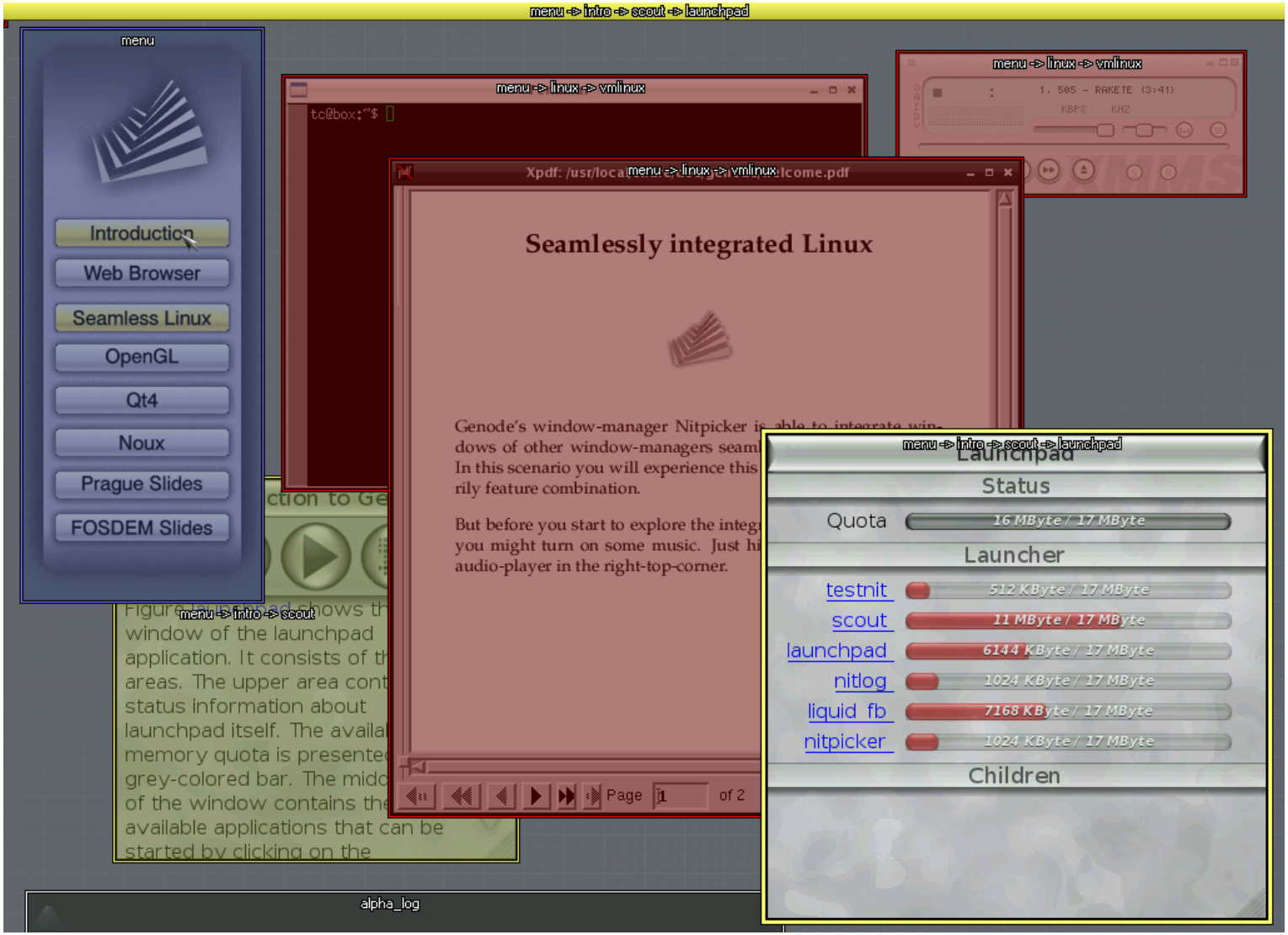
- A Trojan Horse could present an image of a secure window
- Not the secure window must be marked, but all others!

Revoke some degree of freedom from the clients

- Dedicated screen area, reserved for the trusted GUI
- Revoking the ability to use the whole color space

$\rightarrow$ X-Ray mode, activated by special key (x-ray key)

Trusted path (2)



Nitpicker results

Source-code complexity

GUI server	Lines of code
X.org	$>$ 80,000
Trusted X	30,000
DOpE	12,000
EWS	4,500
Nitpicker	$<$ 2,000

- Low performance overhead, no additional copy
- Low-complexity clients are possible (Scout: 4,000 LOC)

Nitpicker results (2)

- Support for legacy software
- Protection against spyware
- Helps to uncover Trojan horses
- Low source-code complexity

→ Poster child of a resource multiplexer

Outline

Interfaces

LOG	Unidirectional debug output
Terminal	Bi-directional input and output *synchronous bulk*
Timer	Facility to block the client
Input	Obtain user input *synchronous bulk*
Framebuffer	Display pixel buffer *synchronous bulk*
PCI	Represents PCI bus, find and obtain PCI devices

Interfaces (2)

ROM Obtain read-only data modules
shared memory

Block Block-device access
packet stream

File_system File-system access
packet stream

NIC Bi-directional transfer of network packets
2 x packet stream

Audio_out Audio output
packet stream

Device drivers

Session type	Location
Timer	`os/src/drivers/timer`

Block	`os/src/drivers/atapi`
	`os/src/drivers/ahci`
	`os/src/drivers/sd_card`
	`dde_linux/src/drivers/usb_drv`

Input	`os/src/drivers/input/ps2` `dde_linux/src/drivers/usb_drv`

Framebuffer	`os/src/drivers/framebuffer/vesa`
	`os/src/drivers/framebuffer/sdl`
	`os/src/drivers/framebuffer/pl11x`
	`os/src/drivers/framebuffer/omap4`

Audio_out	`linux_drivers/src/drivers/audio_out`
Terminal	`os/src/drivers/uart`
NIC	`dde_ipxe/src/drivers/nic` `dde_linux/src/drivers/usb_drv`
PCI	`os/src/drivers/pci`

Resource multiplexers and protocol stacks

Session type	Location
LOG	`os/src/server/terminal_log` `demo/src/server/nitlog`
Framebuffer, Input	`demo/src/server/liquid_framebuffer` `os/src/server/nit_fb`
Nitpicker	`os/src/server/nitpicker`

Terminal	`os/src/server/terminal_crosslink`
	`gems/src/server/terminal`
	`gems/src/server/tcp_terminal`

Resource multiplexers and protocol stacks (2)

Session type	Location
Audio_out	`os/src/server/mixer`
NIC	`os/src/server/nic_bridge`

ROM	`os/src/server/rom_prefetcher`
	`os/src/server/tar_rom`
	`os/src/server/iso9660`

Block	`os/src/server/rom_loopdev`
	`os/src/server/part_blk`
	`gems/src/server/http_block`

File_system	`os/src/server/ram_fs` `libports/src/server/ffat_fs`

Protocol-stack libraries

API	Location

POSIX	`libports/lib/mk/libc.mk`
	`libports/lib/mk/libc_log.mk`
	`libports/lib/mk/libc_fs.mk`
	`libports/lib/mk/libc_rom.mk`
	`libports/lib/mk/libc_lwip.mk`
	`libports/lib/mk/libc_ffat.mk`
	`libports/lib/mk/libc_lock_pipe.mk`
	`libports/lib/mk/libc_terminal.mk`

Qt4	`qt4/lib/mk/qt_*`
OpenGL	`libports/lib/mk/gallium.mk`

Runtime environments

Runtime	Location
Init	`os/src/init`
Loader	`os/src/server/loader`
L4Linux	`ports-foc/src/l4linux`
L4Android	`ports-foc/src/l4android`
OKLinux	`ports-okl4/src/oklinux`
Vancouver	`ports/src/vancouver`
Noux	`ports/src/noux`
GDB Monitor	`ports/src/app/gdb_monitor`
Python	`libports/lib/mk/x86_32/python.mk`
Lua	`libports/lib/mk/moon.mk`

Thank you

Genode OS Framework

http://genode.org

Genode Labs GmbH

http://www.genode-labs.com

Source code at GitHub

http://github.com/genodelabs/genode