Genode OS Framework - Compositions

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

Outline

1. Virtualization techniques
2. Enslaving services
3. Dynamic workloads
4. Current ventures

Outline

1. Virtualization techniques
2. Enslaving services
3. Dynamic workloads
4. Current ventures

Virtualization techniques

Flavors

Faithful → virtual hardware platform

Para → modified guest OS kernel

OS-level → source-level user-land compatibility

Process-level → tailored process environment

C-runtime → tailored runtime within process

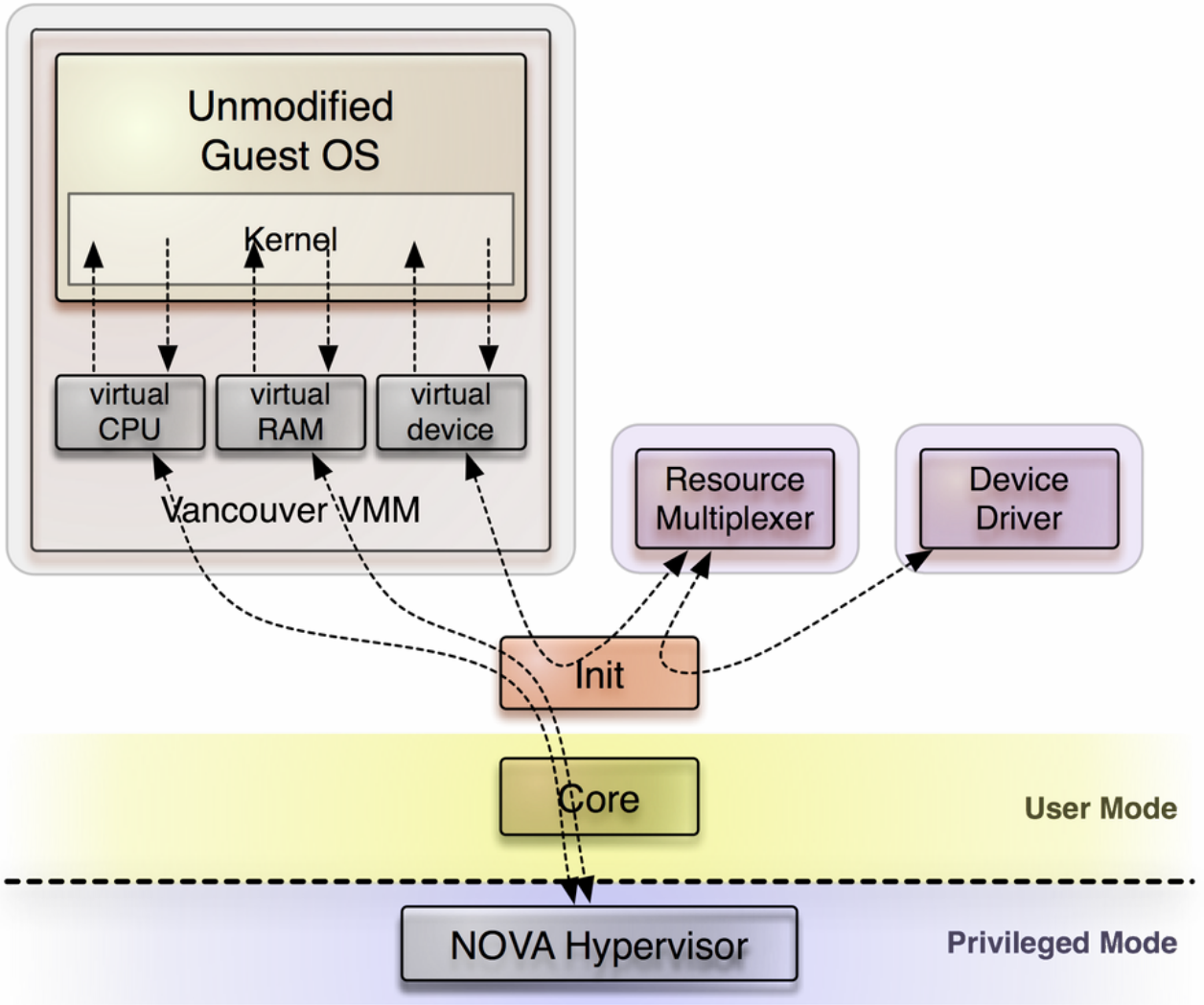
Faithful virtualization

- Requires CPU and kernel with virtualization support
  → Intel VT, AMD SVM, ARM Cortex-A15
  → NOVA, Fiasco.OC (not supported yet)

- VMM emulates complete platform (CPU+memory+devices)

- No guest OS modifications required

Faithful virtualization (2)



Faithful virtualization (3)

```
<config>
  <machine>
    <mem start="0x0" end="0xa0000"/>
    <mem start="0x100000" end="0x2000000"/>
    <nullio io_base="0x80" />
    <pic io_base="0x20" elcr_base="0x4d0"/>
    <pic io_base="0xa0" irq="2" elcr_base="0x4d1"/>
    <pit io_base="0x40" irq="0"/>
    <scp io_port_a="0x92" io_port_b="0x61"/>
    <kbc io_base="0x60" irq_kbd="1" irq_aux="12"/>
    <keyb ps2_port="0" host_keyboard="0x10000"/>
    <mouse ps2_port="1" host_mouse="0x10001"/>
    <rtc io_base="0x70" irq="8"/>
    <serial io_base="0x3f8" irq="0x4" host_serial="0x4711"/>
    <hostsink host_dev="0x4712" buffer="80"/>
    <vga io_base="0x03c0"/>
    <vbios_disk/> <vbios_keyboard/> <vbios_mem/>
    <vbios_time/> <vbios_reset/> <vbios_multiboot/>
    <msi/> <ioapic/>
    <pcihostbridge bus_num="0" bus_count="0x10" io_base="0xcf8" mem_base="0xe0000000"/>
    <pmtimer io_port="0x8000"/>
    <vcpu/> <halifax/> <vbios/> <lapic/>
  </machine>
  <multiboot>
    <rom name="bootstrap"/>  <rom name="fiasco"/>
    <rom name="sigma0.foc"/> <rom name="core.foc"/>
  </multiboot>
</config>
```

Paravirtualization

- No virtualization support needed

  → Can be used on current ARM platforms

- Guest OS ported to virtual platform

- Binary compatible to guest OS userland

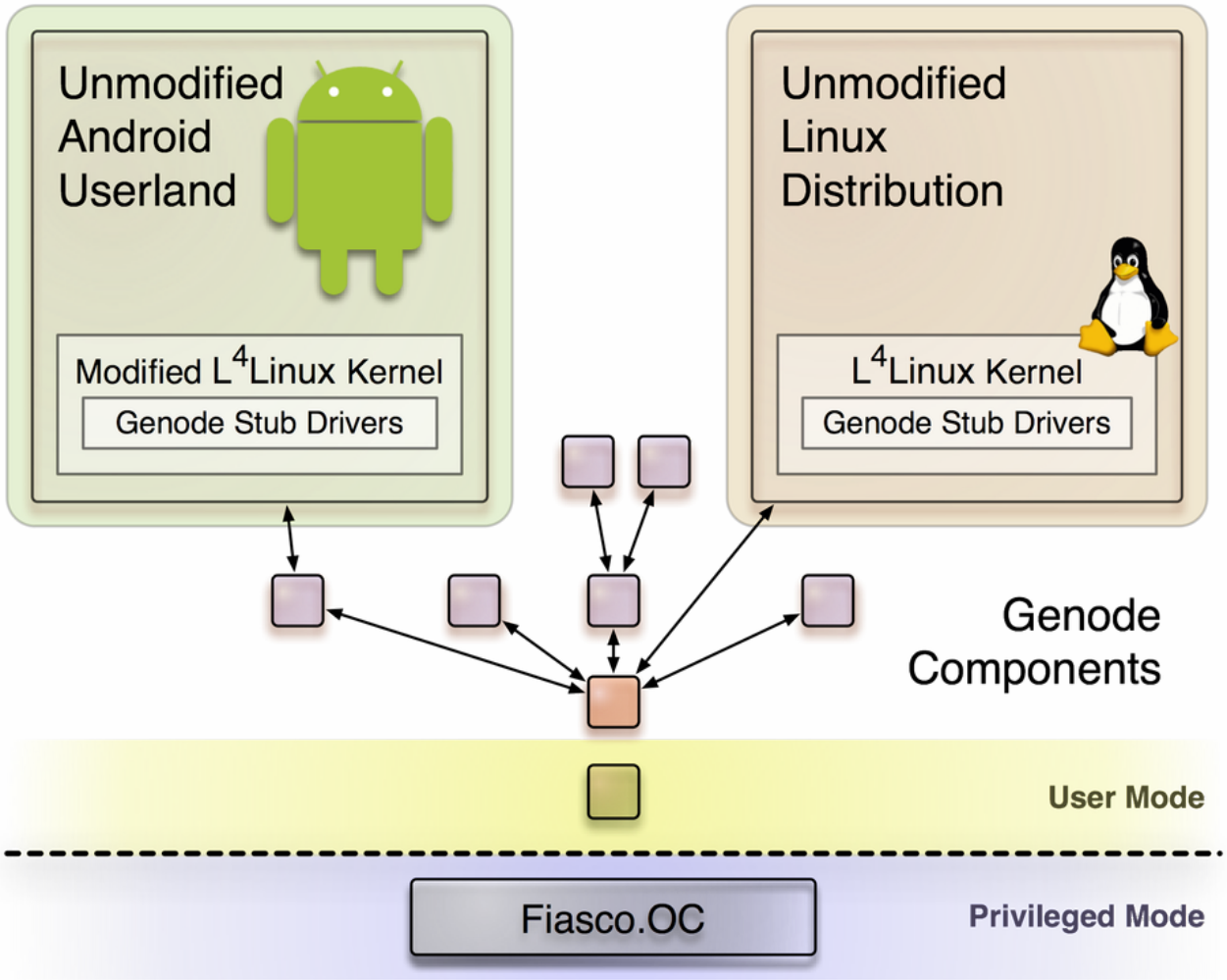
- Guest OS kernel must be modified

  → Solution inherently base-platform specific

  L4Linux for Fiasco.OC
  OKLinux for OKL4

  → Ongoing maintenance work

Paravirtualization (2)



Paravirtualization (3)

Stub drivers

Terminal	→	character device
Block	→	block device
NIC	→	net device
Framebuffer	→	`/dev/fb*` device
Input	→	`/dev/input/*` device
Nitpicker	→	`ioctl` extension to `/dev/fb*` *(OKLinux only)*
Audio_out	→	ALSA *(OKLinux only)*

no direct hardware access

OS-level virtualization

Idea: Provide Unix kernel interface as a service

fundamentals

- `write`, `read`
- `stat`, `lstat`, `fstat`, `fcntl`
- `ioctl`
- `open`, `close`, `lseek`
- `dirent`
- `getcwd`, `fchdir`
- `select`
- `execve`, `fork`, `wait4`
- `getpid`
- `pipe`
- `dup2`
- `unlink`, `rename`, `mkdir`

networking

- `socket`
- `getsockopt`, `setsockopt`
- `accept`
- `bind`
- `listen`
- `send`, `sendto`
- `recv`, `recvfrom`
- `getpeername`
- `shutdown`
- `connect`
- `getaddrinfo`

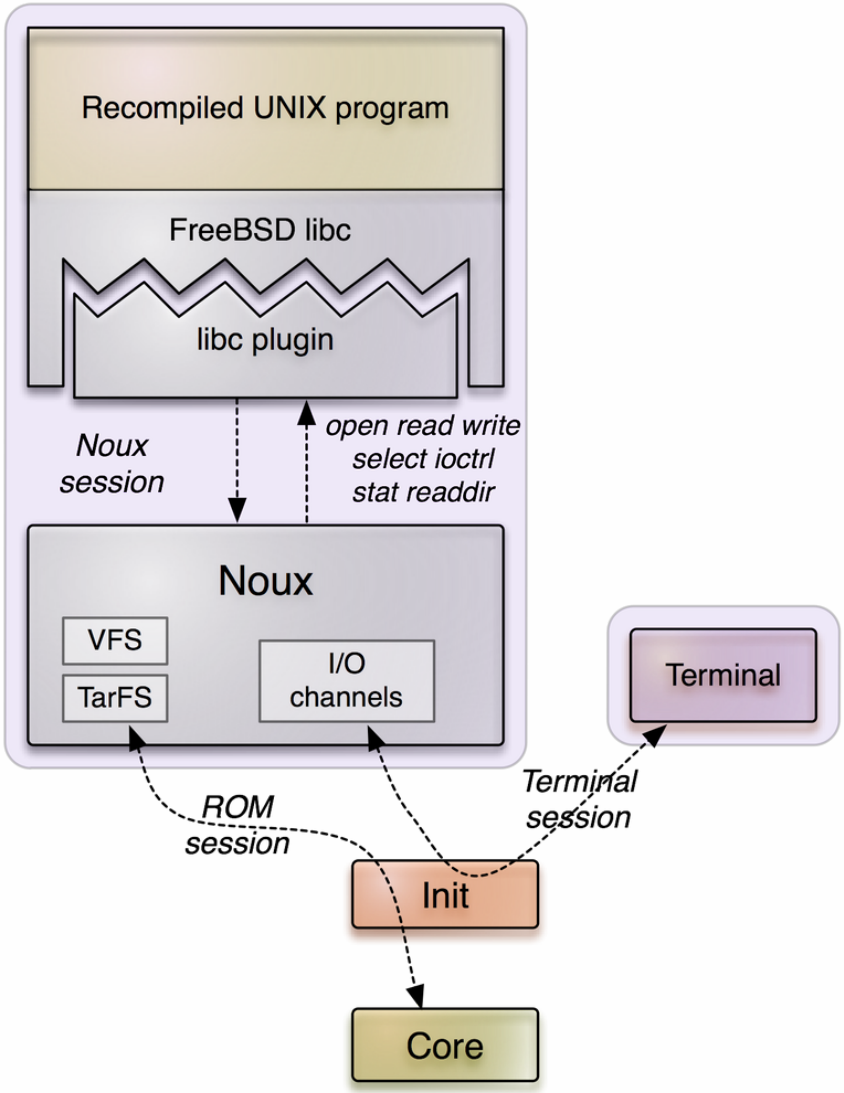
In contrast, Linux has more than 300 syscalls

OS-level virtualization (2)

Things we don't need to consider

- Interaction with device drivers
- Unix initialization sequence
- Users, groups
  Instance never shared by multiple users
  The opposite: One user may run many instances
- Multi-threading
- Scalability of a single instance
  Each instance serves one specific (limited) purpose
  Run many instances in order to scale!

OS-level virtualization (3)



Noux: Running VIM

noux config

```
<config>
  <fstab> <tar name="vim.tar" /> </fstab>
  <start name="/bin/vim">
    <env name="TERM" value="linux" />
    <arg value="--noplugin" />
    <arg value="-n" /> <!-- no swap file -->
    <arg value="-N" /> <!-- no-compatible mode -->
  </start>
</config>
```

Noux: Bash + file system

noux config

```
<config>
  <fstab>
    <tar name="coreutils.tar" />
    <tar name="vim.tar" />
    <tar name="bash.tar" />
    <dir name="home"> <fs label="home" /> </dir>
    <dir name="ram">  <fs label="root" /> </dir>
    <dir name="tmp">  <fs label="tmp" />  </dir>
  </fstab>
  <start name="/bin/bash">
    <env name="TERM" value="linux" />
  </start>
</config>
```

Noux: Bash + file system (2)

ram_fs config

```
<config>
  <content>
    <dir name="tmp">
      <rom name="init" as="something" />
    </dir>
    <dir name="home">
      <dir name="user">
        <rom name="timer" />
      </dir>
    </dir>
  </content>
  <policy label="noux -> root" root="/" />
  <policy label="noux -> home" root="/home/user" writeable="yes" />
  <policy label="noux -> tmp"  root="/tmp"       writeable="yes" />
</config>
```

Noux features

- Executes unmodified GNU software
  Bash, VIM, GCC, Coreutils, Lynx, GDB...
- Supports stacked file systems
- Instance starts in fraction of a second
- Uses original GNU build system → Porting is easy
- Two versions
  - `noux/minimal`
  - `noux/net` (includes TCP/IP)

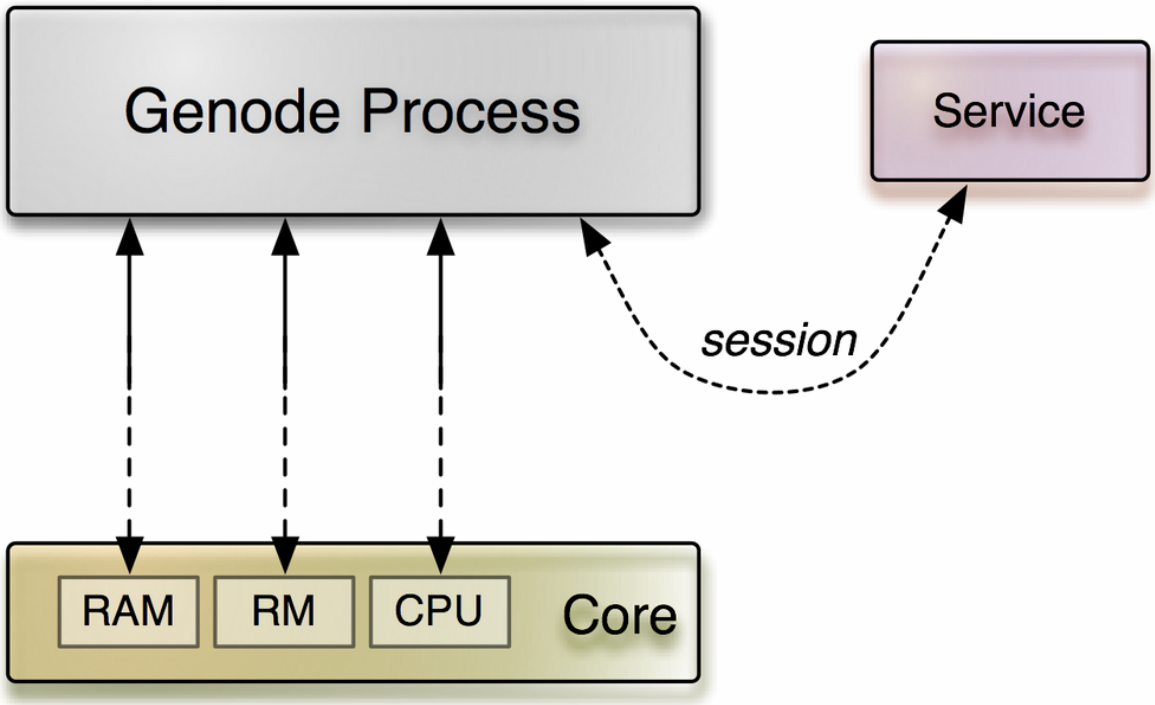
less than 5,000 LOC

Process-level virtualization

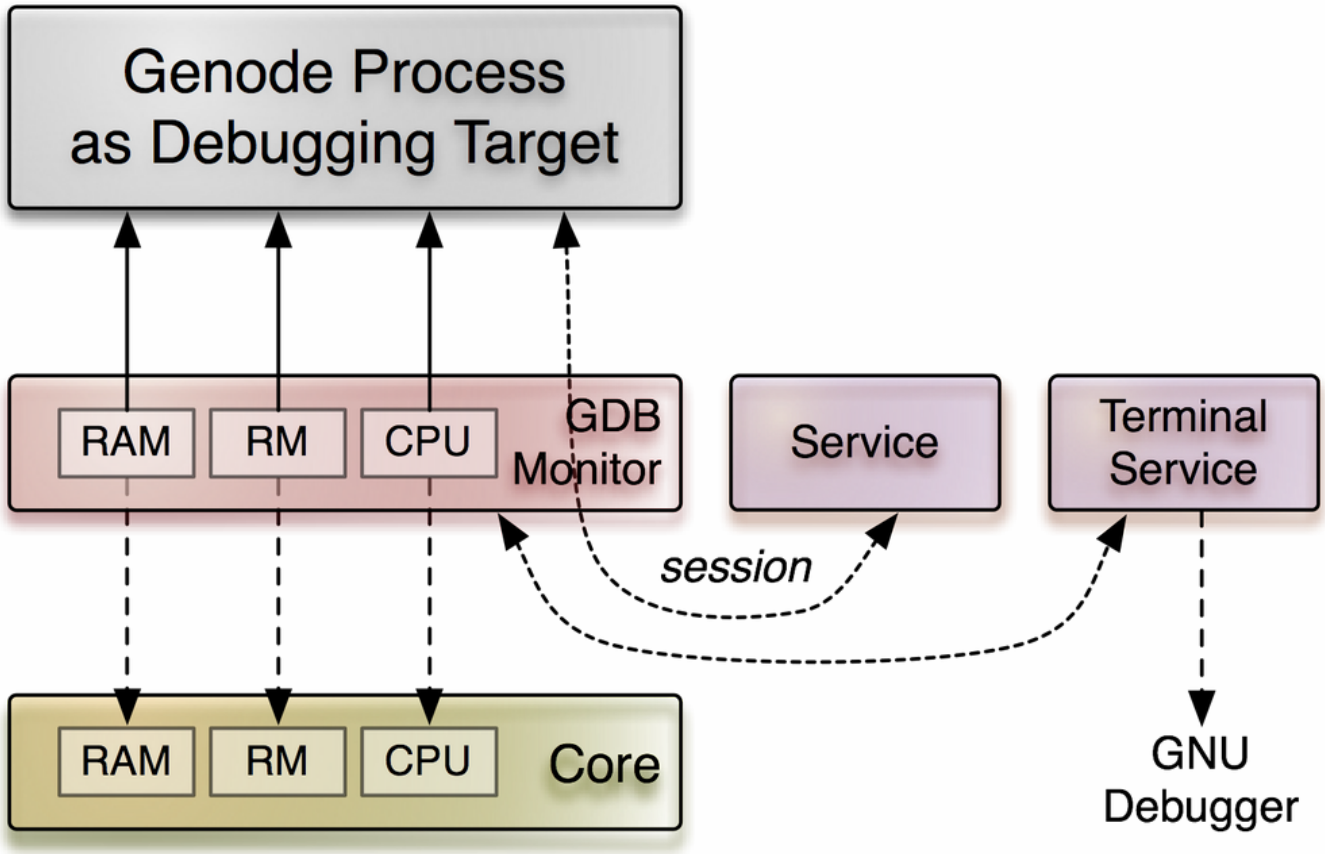
Opportunity: Virtualization of individual session interfaces

- Monitoring of session requests
- Customization of existing services
  → Reuse of existing components
  → Separation of policy and mechanisms

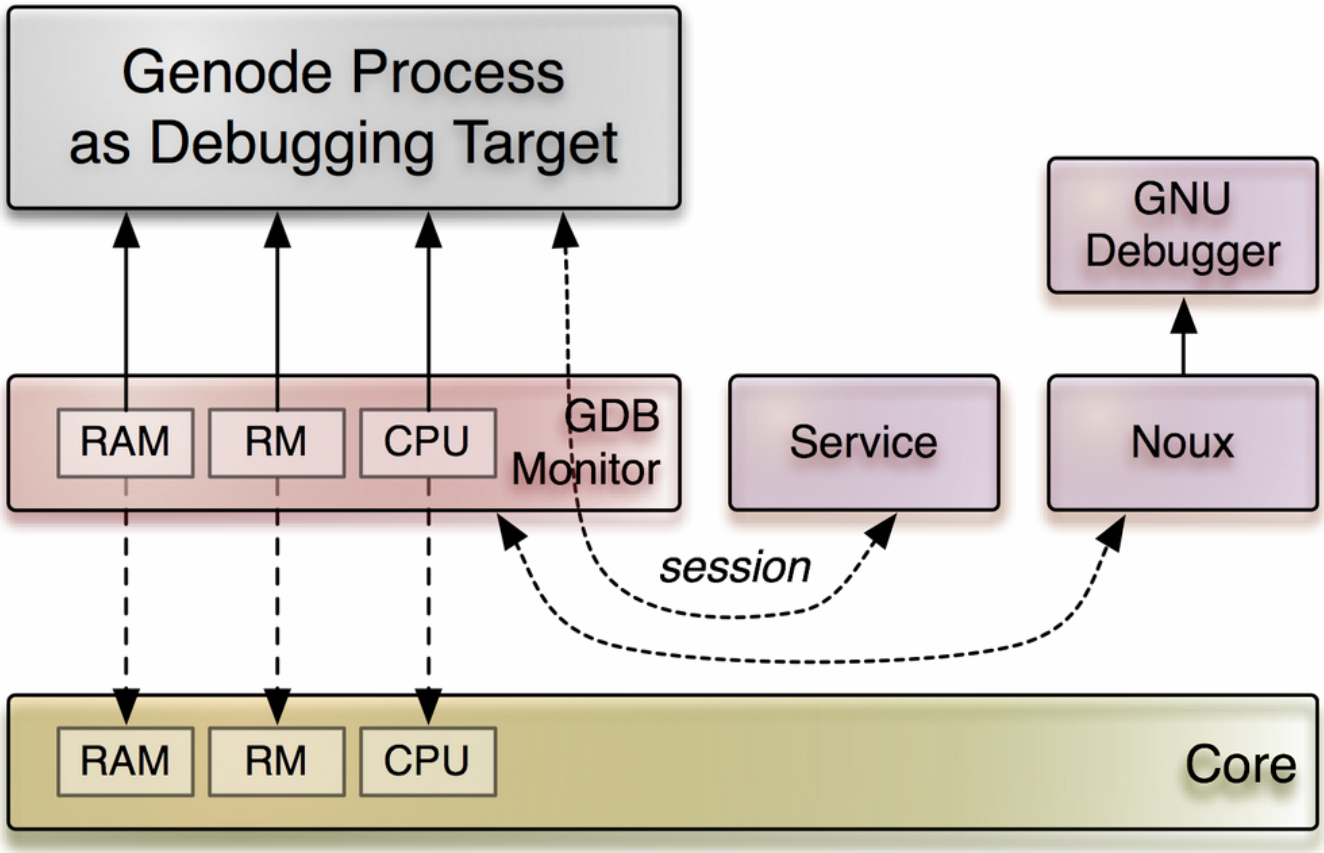
Process-level virtualization (2)



Process-level virtualization (3)



Process-level virtualization (4)



GDB monitor features

- Supported on Fiasco.OC and OKL4
- Break-in by user or segfault
- Source-level debugging
- Backtraces
- Breakpoints
- Single-stepping
- Debugging of multi-threaded processes
- Debugging of dynamically linked binaries

C-runtime customization

FreeBSD libc turned into modular C runtime

`libports/lib/mk/libc.mk`

`libports/lib/mk/libc_log.mk`

`libports/lib/mk/libc_fs.mk`

`libports/lib/mk/libc_rom.mk`

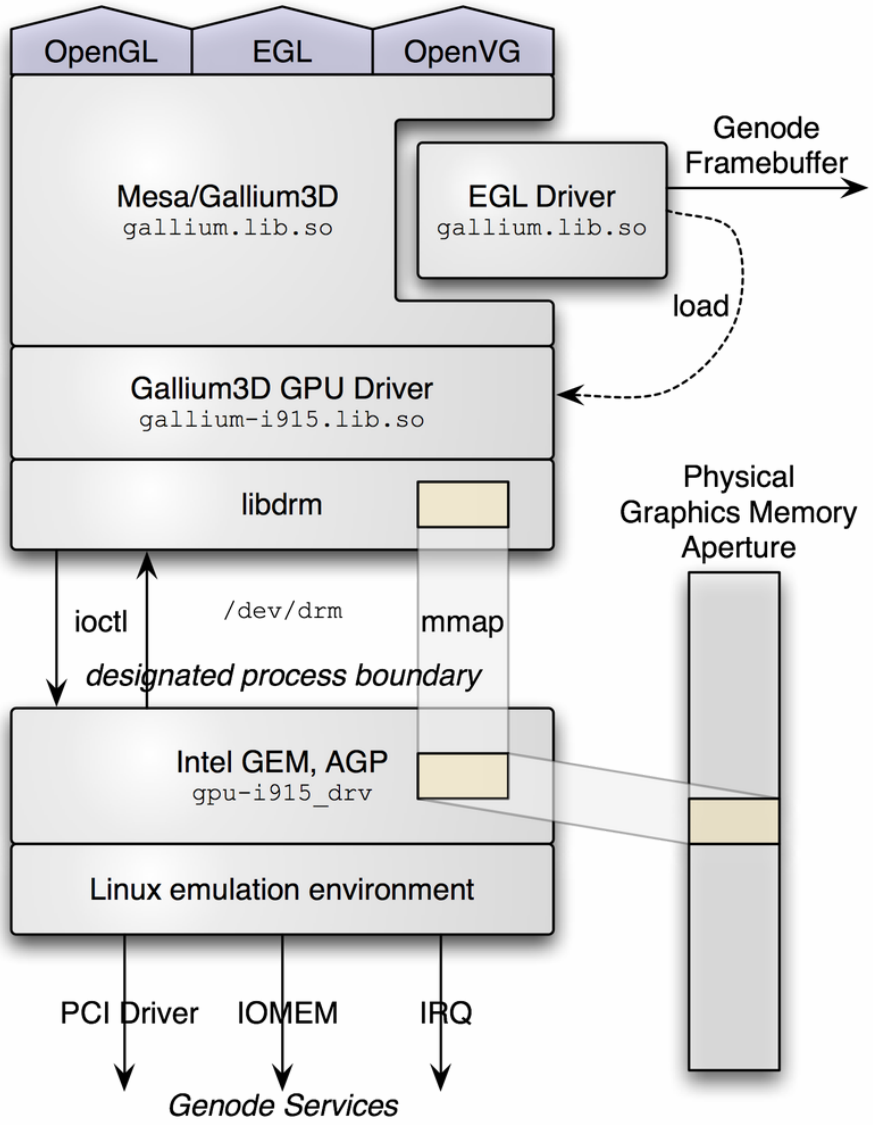
`libports/lib/mk/libc_lwip.mk`

`libports/lib/mk/libc_ffat.mk`

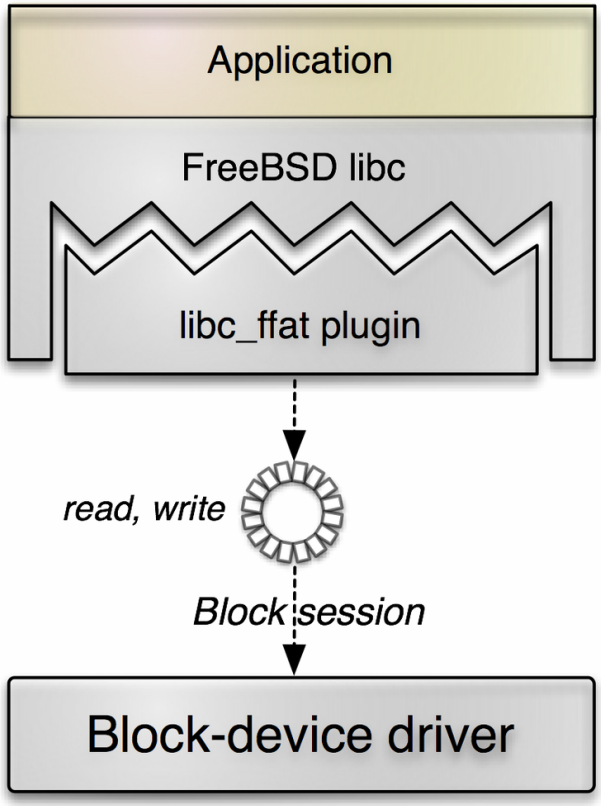
`libports/lib/mk/libc_lock_pipe.mk`

→ *application-specific plugins*

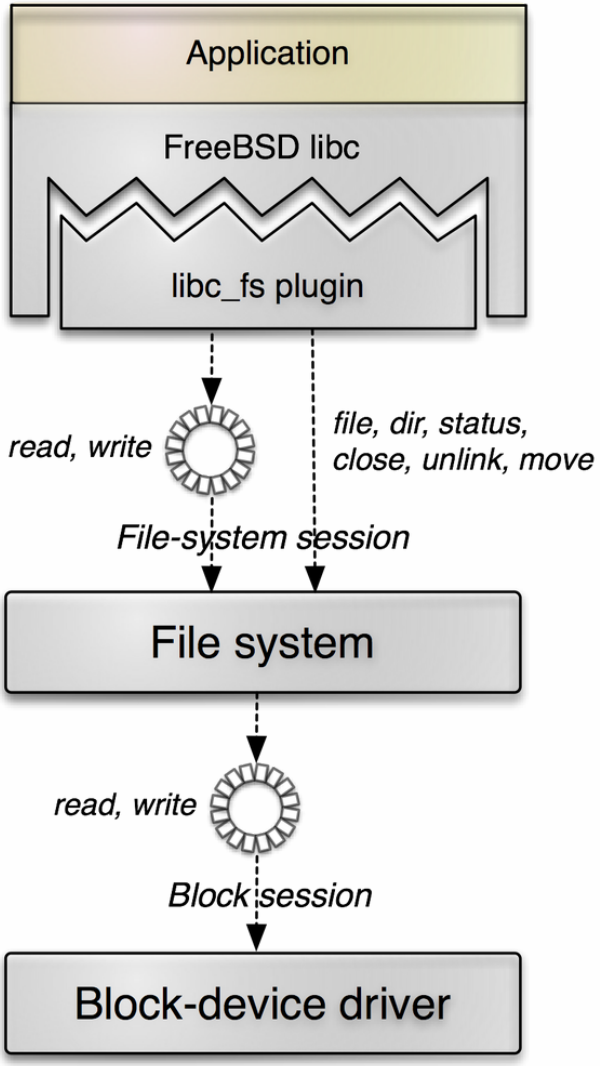
C-runtime customization example



C-runtime customization example (2)



C-runtime customization example (3)



Outline

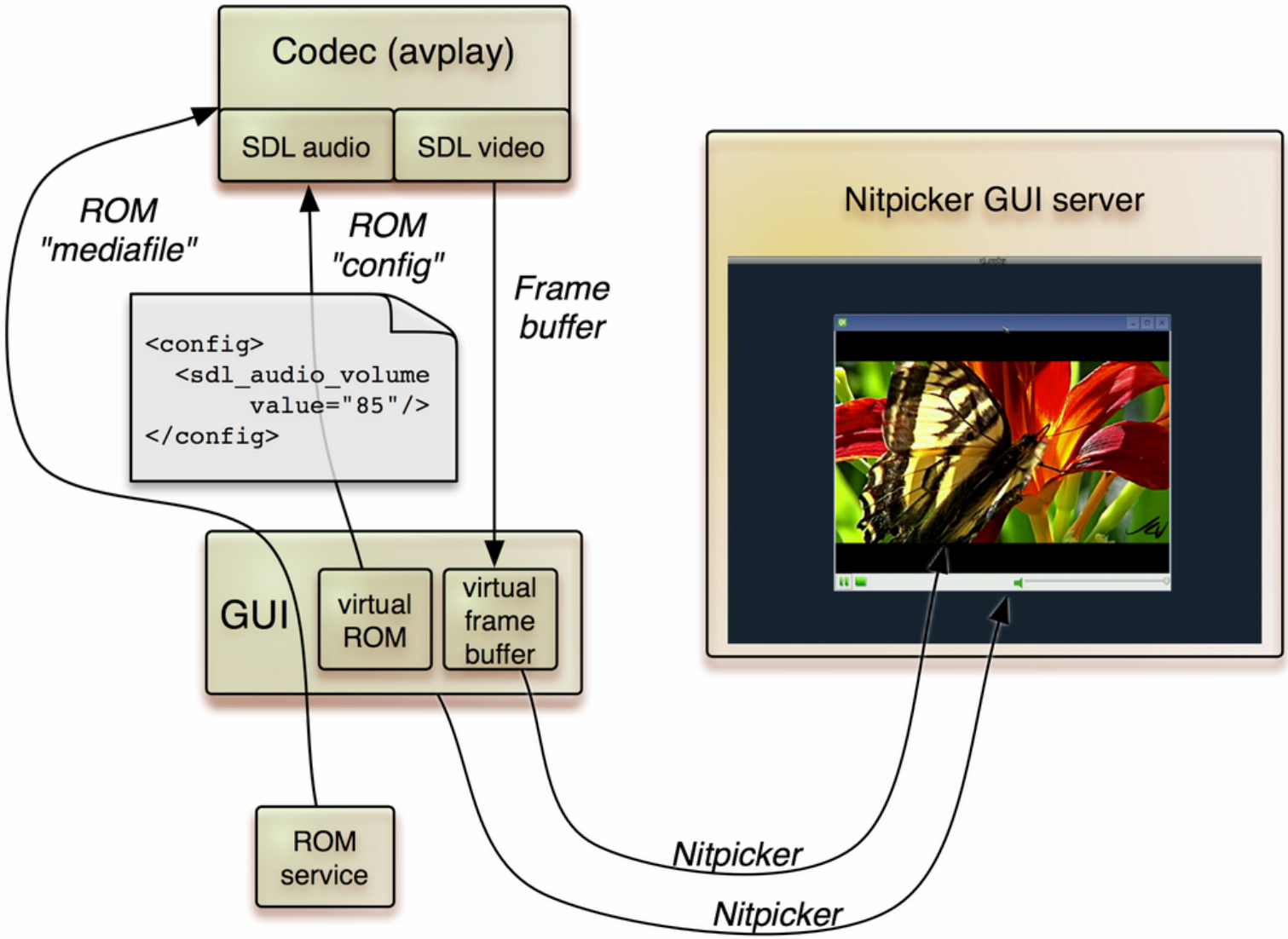
1. Virtualization techniques
2. Enslaving services
3. Dynamic workloads
4. Current ventures

Enslaving services

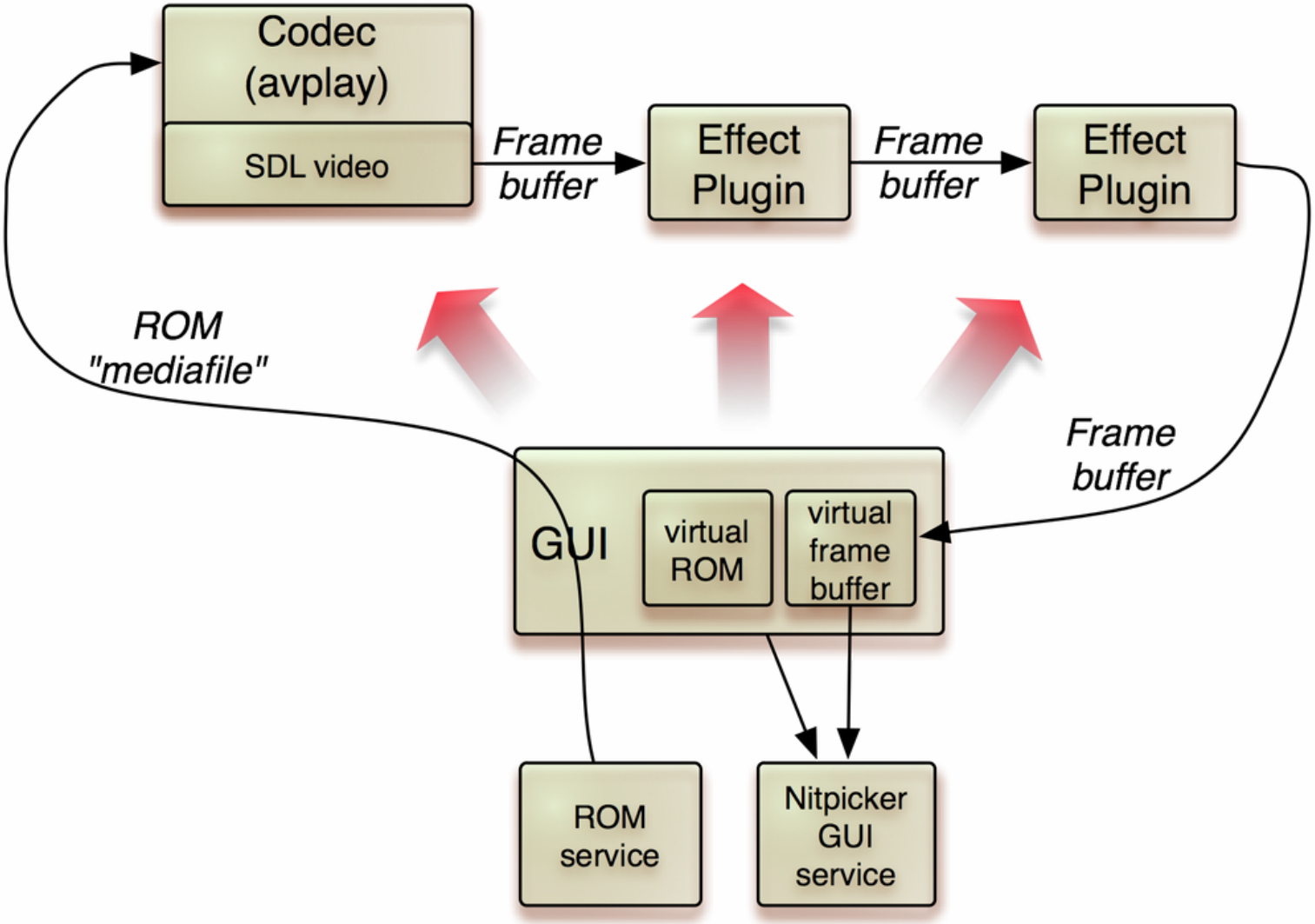
Idea: Run a service as a child

- Sandboxing
- Easy code reuse
- Plugin mechanism

Media player



Media player (2)



Slave API helper

```
#include <os/slave.h>
...
struct Policy : Slave_policy
{
  char const **_permitted_services() const
  {
    static char const *permitted_services[] = { "CAP", "RM", "RAM", "LOG", 0 };
    return permitted_services;
  };

  Policy(Rpc_entrypoint &slave_ep) : Slave_policy("ram_fs", slave_ep) { }

  bool announce_service(const char     *service_name,
                        Root_capability root,
                        Allocator      *alloc,
                        Server         *server)
  {
    /* policy goes here */
    ...
  }
};

...

Rpc_entrypoint ep(&cap, STACK_SIZE, "slave_ep");
Policy         policy(ep);
Slave          slave(ep, policy, RAM_QUOTA);
```

Outline

1. Virtualization techniques
2. Enslaving services
3. Dynamic workloads
4. Current ventures

Challenges of dynamic systems

Typical problems of dynamic systems

- Clean revocation of resources
- Run-time discovery
- Dynamic policies
- Run-time adaptive components

Orderly destruction of subsystems

Challenges

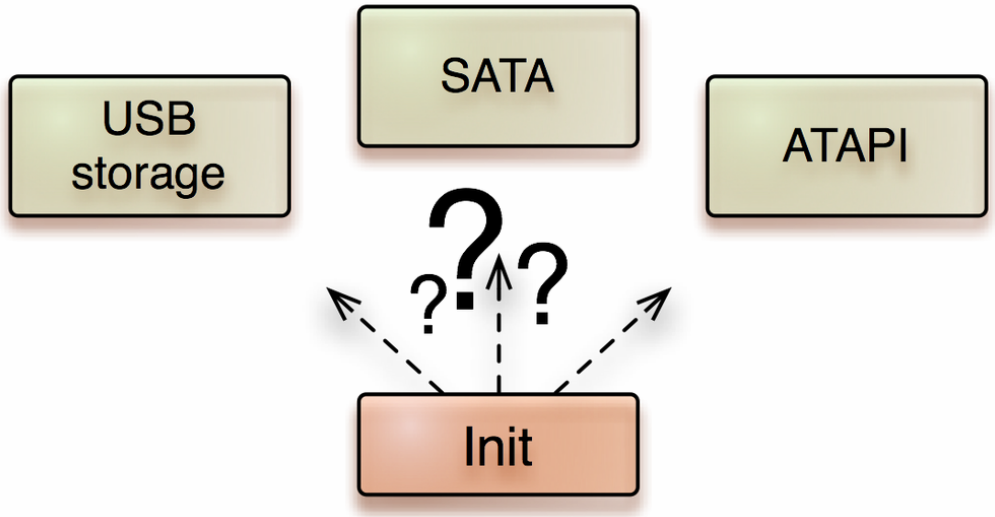
- Resource leaks
- Dangling references
- Locked resources
- Used servers need cleanup

Genode comes with simple solution

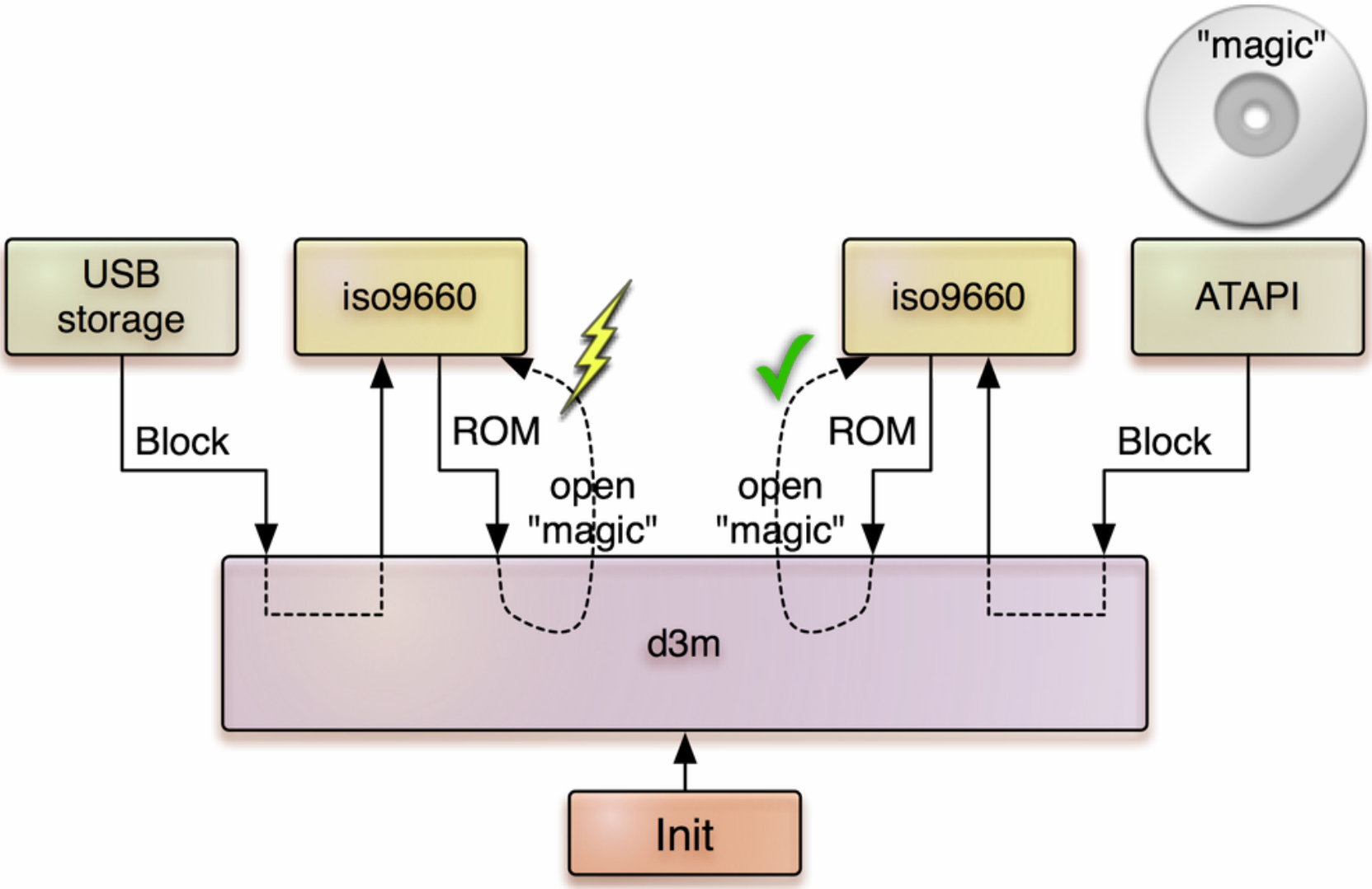
Parent closes all sessions in reverse order

→ *Server side: looks like client closes session*
→ *Resource donations are reverted*
→ *Works for arbitrarily complex subsystems*

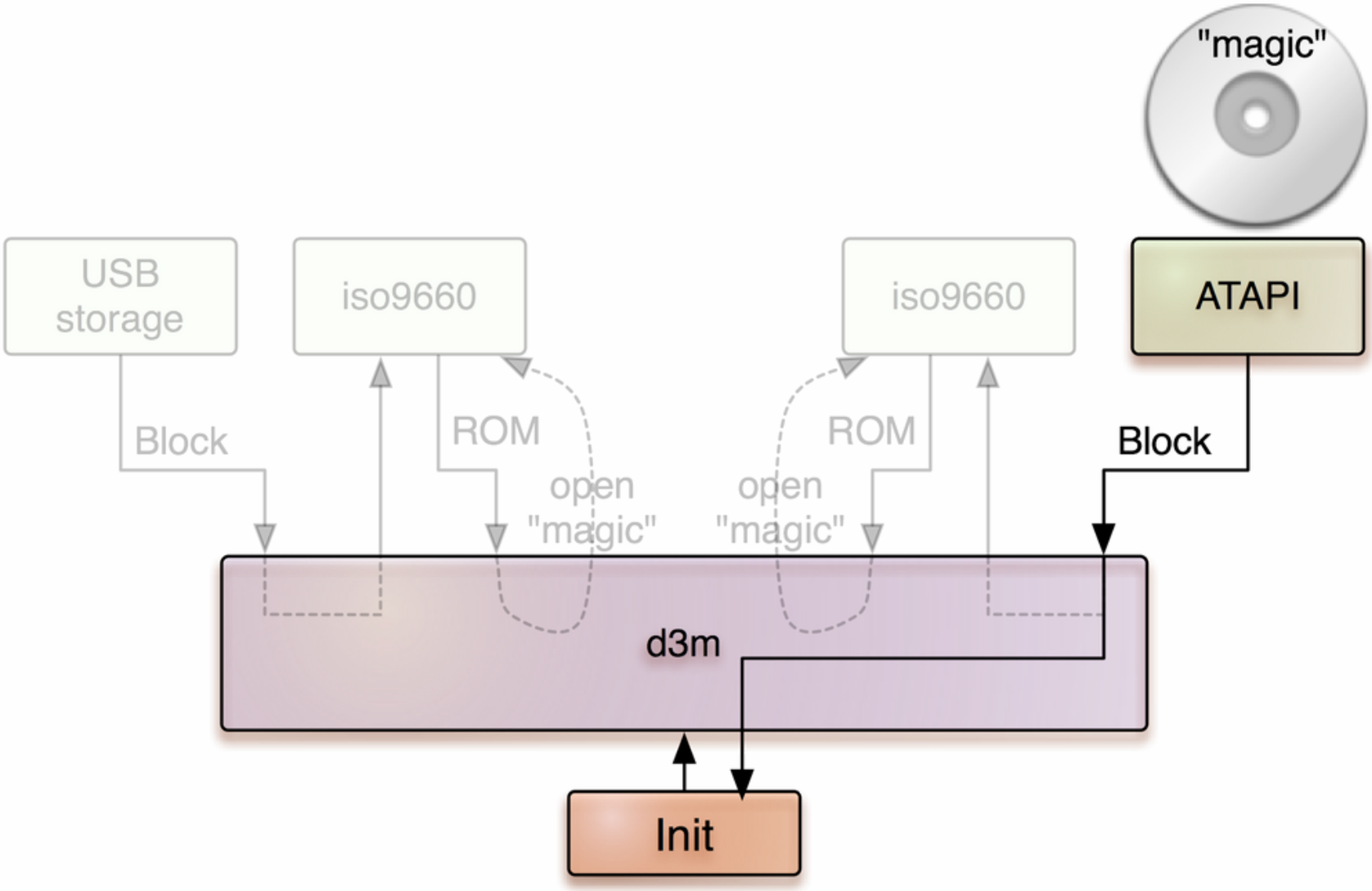
Boot medium detection



Boot medium detection (2)



Boot medium detection (3)



Dynamic system configuration

Problems

- Change screen resolution at runtime
- Audio-mixing parameters
- Touchscreen calibration
- Resizing terminal windows
- Policy for hot-plugged device resources

Dynamic system configuration (2)

Straight-forward approach

Introduce problem-specific RPC interfaces

Disadvantages

- New RPC interfaces $\rightarrow$ added complexity
- Specific to the server *implementation*
- Redundancy to existing (static) configuration concept

Dynamic system configuration (3)

Generalized solution

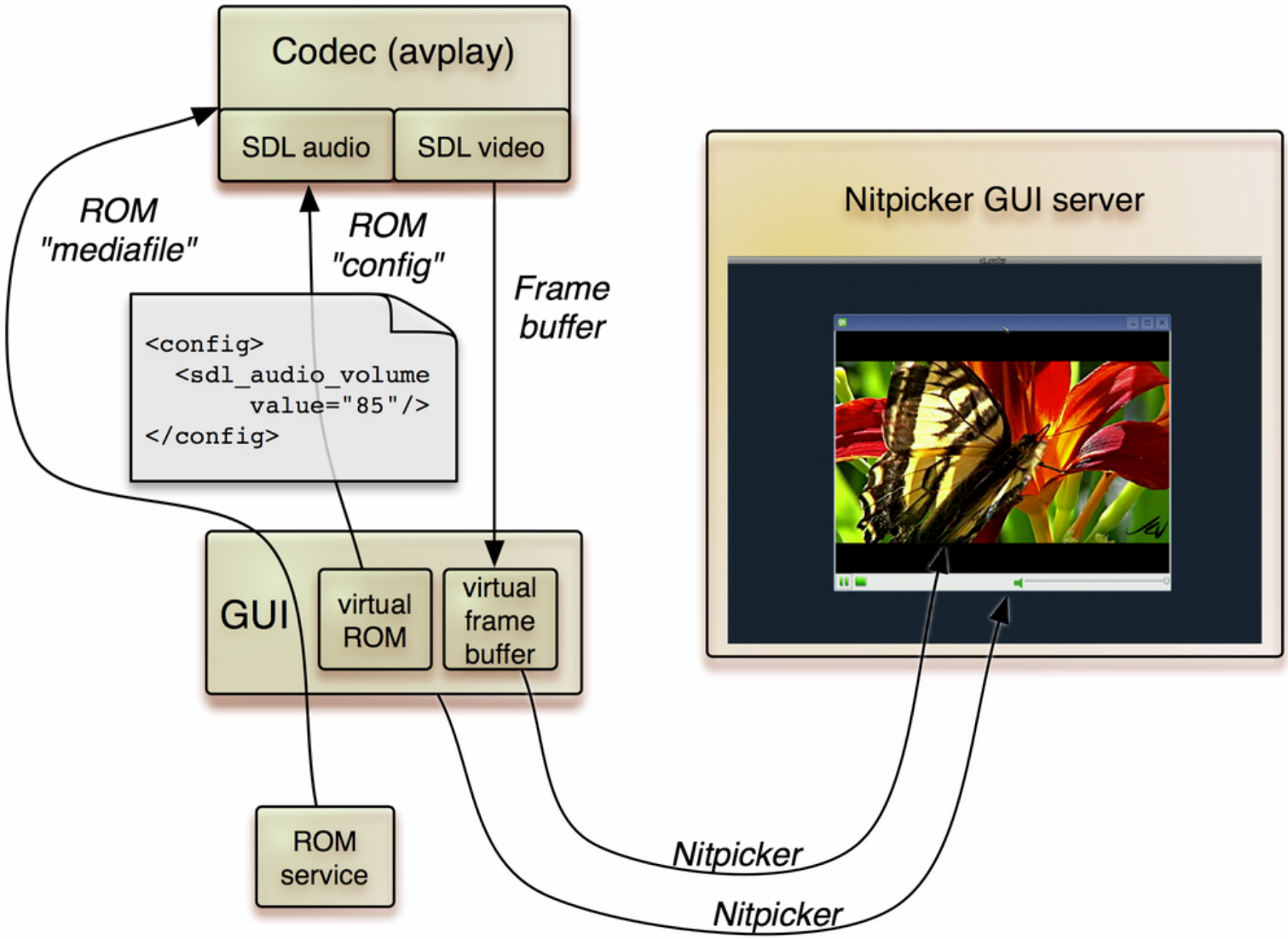
- Turn static config mechanism into dynamic mechanism

How?

- Add single RPC function to ROM session interface:

  `void sigh(Signal_context_capability sigh)`
- Client responds to signal by re-acquiring session resources

Dynamic system configuration (4)



Adaptable session interfaces

Pattern

- Install signal handler
- Respond to session-update signals
- Change mode of operation
- Transactional semantics

→ *works for ROM, framebuffer, terminal*

Loader service

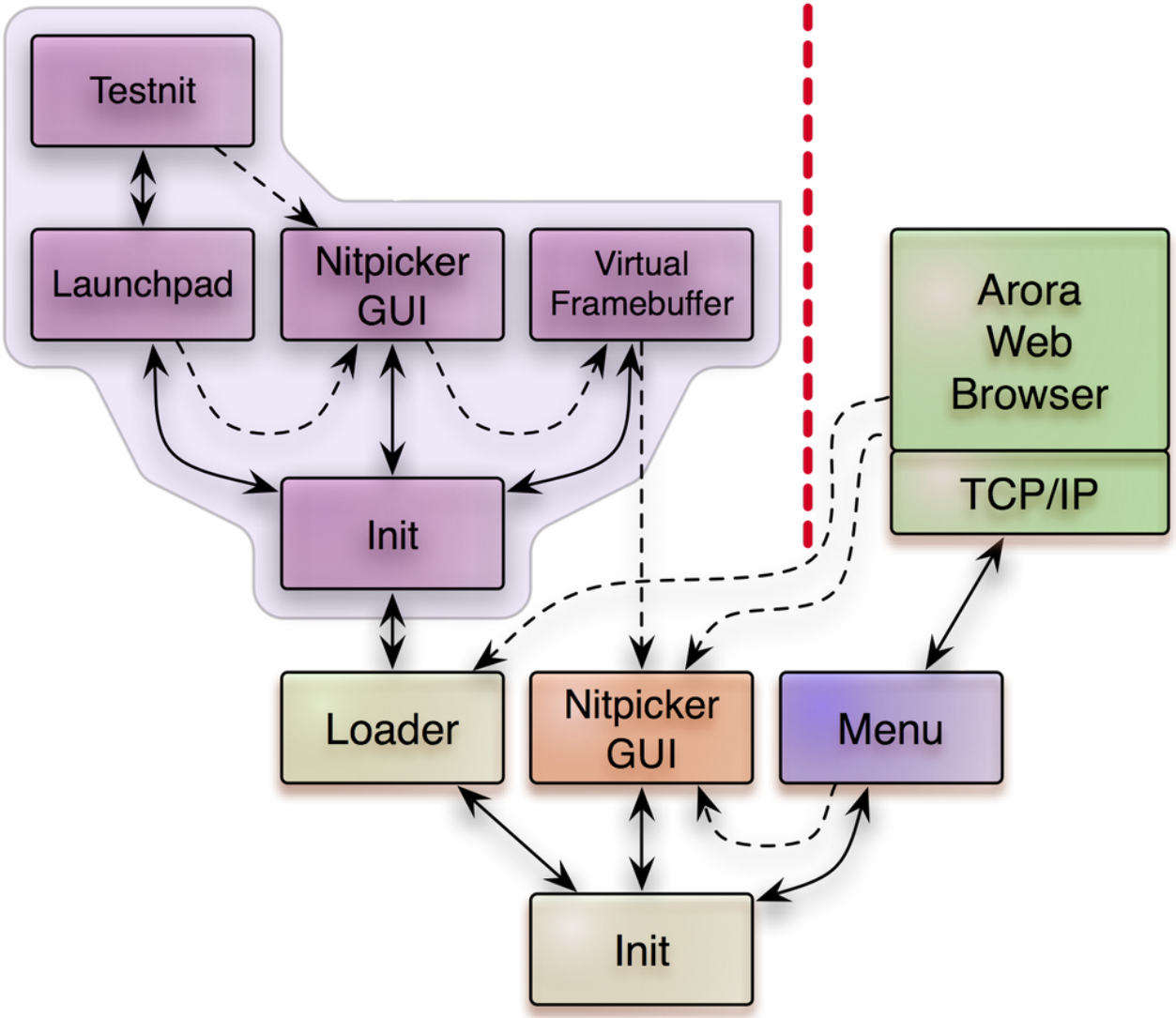
Challenges

- Start and stop subsystems at runtime
- Controlled by software
- Decouple started subsystem from controlling software

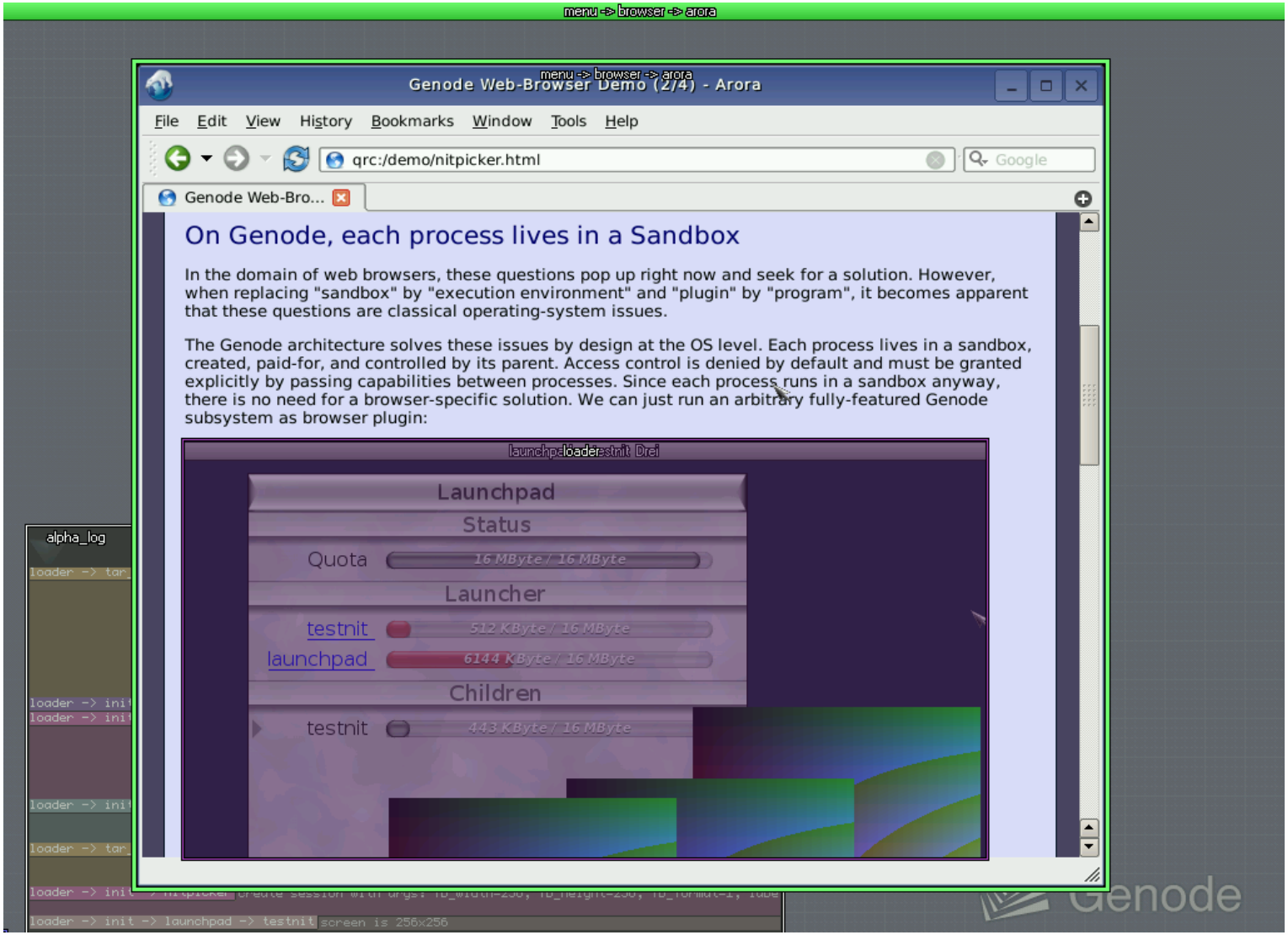
Solution

- Trusted *loader* service
- Client pays
- Client configures subsystem
- Client cannot interfere during runtime

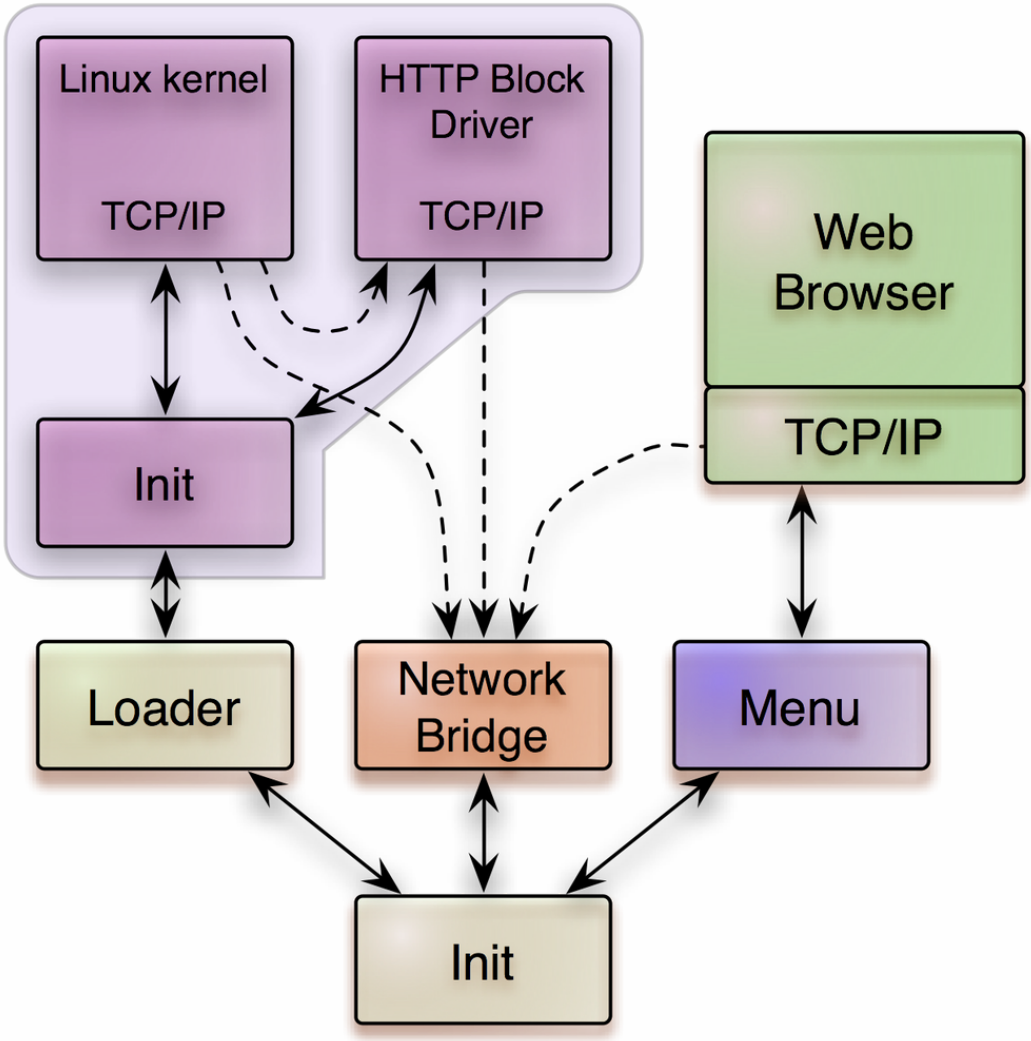
Loader service



Loader service (2)



Loader service (3)



Outline

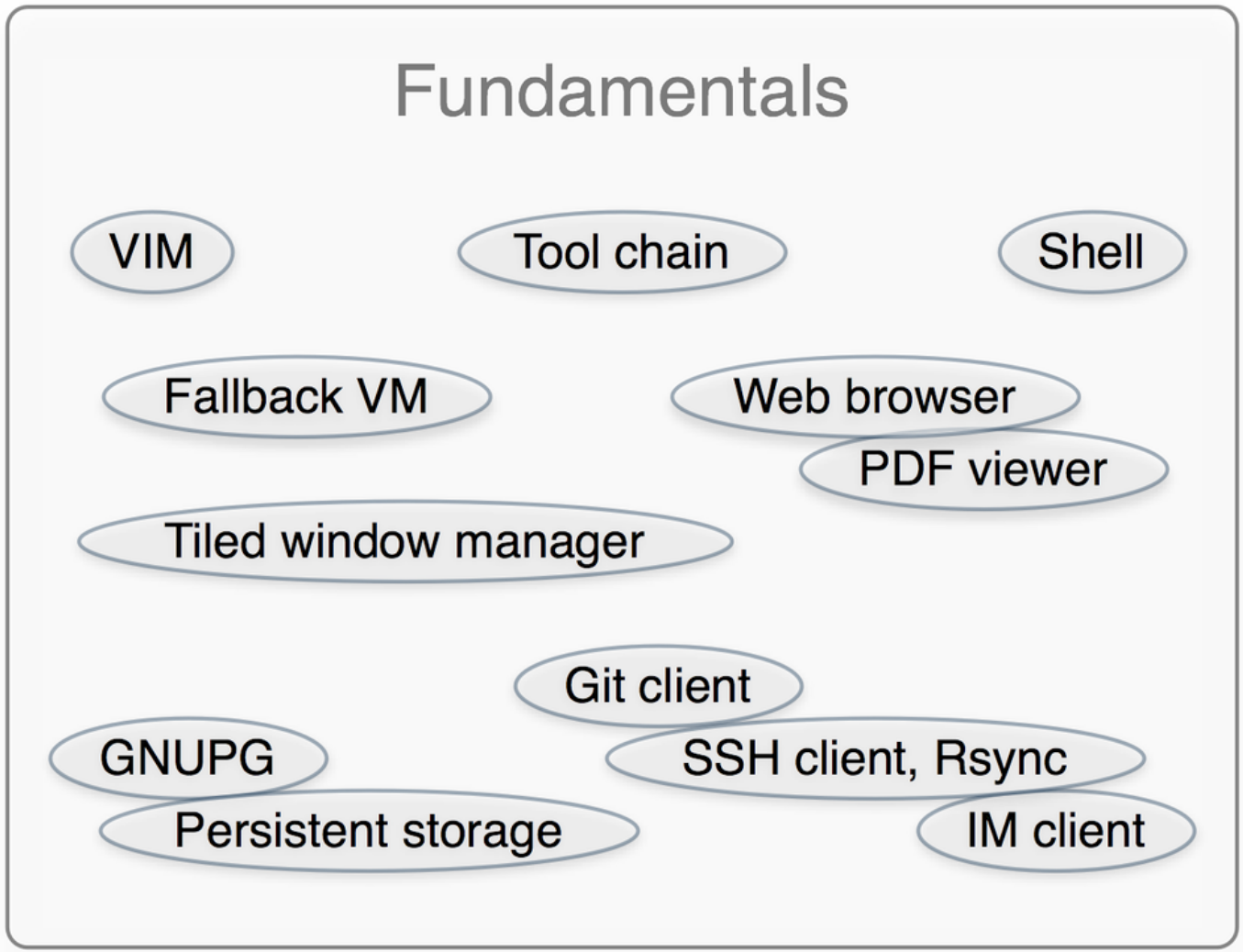
1. Virtualization techniques
2. Enslaving services
3. Dynamic workloads
4. Current ventures

Big picture

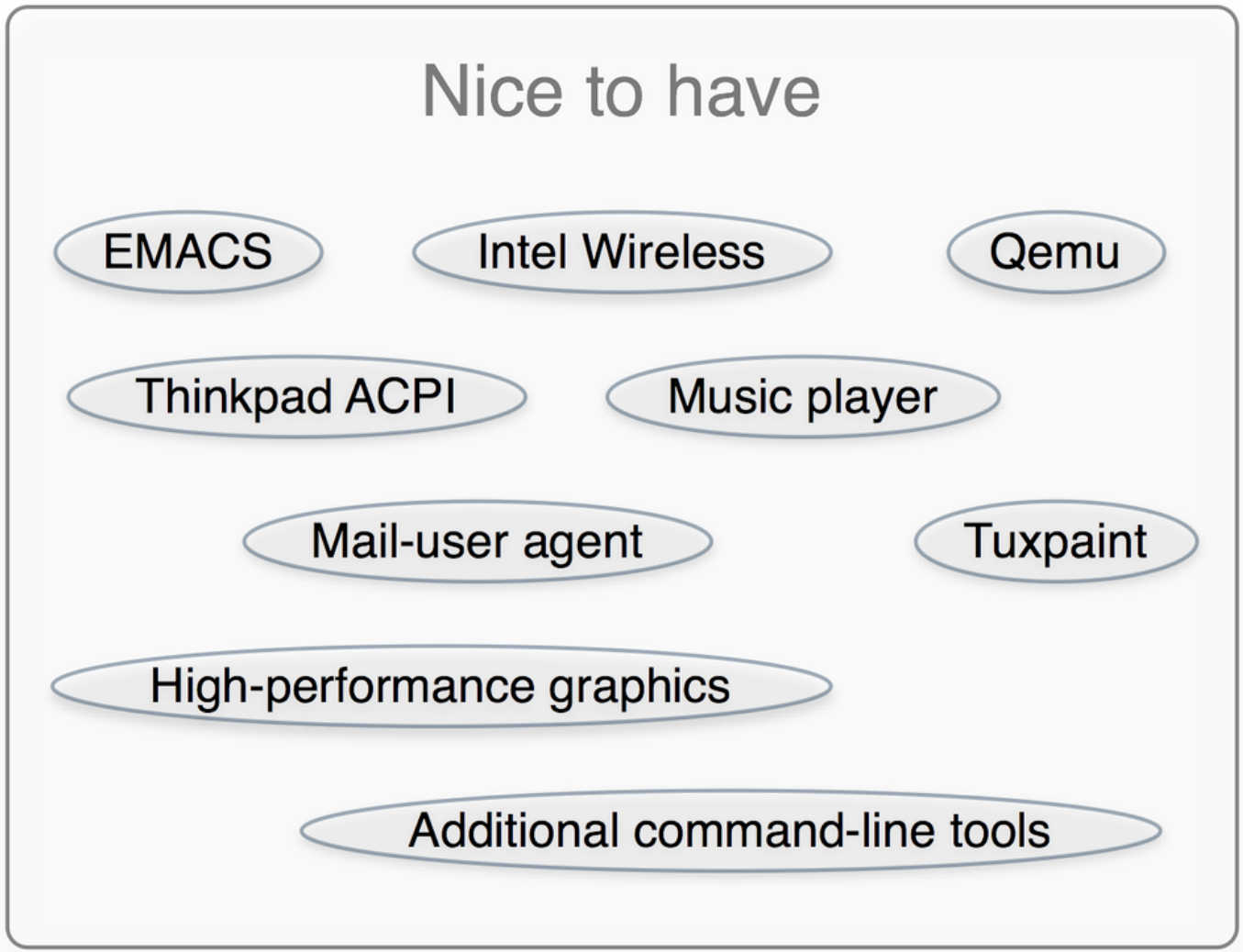
Eating our own dog food

Using Genode as our primary OS by the end of 2012

Big picture (2)



Big picture (3)



Live CD

Present vision of Genode as general-purpose OS

Scenarios:

- Webkit-based web browser
- Media replay
- (Para-)virtualized Linux
- Running the tool chain
- On-target debugging using GDB

Genode on raw hardware

Typical base platform

kernel ($>$ 10,000 LOC) + core (10,000 LOC)

$\rightarrow$ *TCB $>$ 20,000 LOC*

Idea: Merge kernel and core

- Reduce redundant data structures
  $\rightarrow$ No in-kernel mapping data base
  $\rightarrow$ No memory allocation in kernel
- Simplify interaction of kernel $\leftrightarrow$ roottask
- Solve kernel resource management problem

$\rightarrow$ *Core on bare machine (ARMv7): 13,000 LOC*

Self-hosting

Goal: Compile Genode on Genode

Approach

- Noux runtime
- Use unmodified build system

What is working

GNU GCC, binutils, bash, coreutils, make

Current topics

- Corner cases of POSIX API
- Stability
- Performance

Noux: Unix networking tools

Needed command-line tools

- netcat, wget, ...
- Lynx + SSL
- SSH

Approach

Integrate lwIP into Noux runtime

→ *One TCP/IP stack per Noux instances*

File systems

Current implementations

- In-memory file system (ram_fs)
- VFAT file system (ffat_fs)

Desired

- Transactional file system
- Advanced block allocation (avoiding fragmentation)
- Compatibility to Linux

Vancouver virtual machine monitor

Faithful virtualization on NOVA

- Runs Linux as guest at near-native performance

Current state on Genode

- Bootstaps Fiasco.OC and Pistachio kernels
- No interrupts

Working topics

- Complement port with interrupts
- Booting Linux
- Integration with Genode session interfaces
  NIC, Block, Framebuffer, Input

Genode.org hosted on Genode

Ingredients

- Static genode.org website
- lwIP
- NIC drivers

Options

- Custom web server
- Web server ported via libc + libc_lwip
- Web server running as Noux process

Linux - Capabilities via SCM rights

Idea

- File descriptors are process-local names
  capability
- Unix domain sockets can carry file descriptors
  capability delegation
- Messages can be sent to file descriptors
  capability invocation
- Run sub system within chroot environment

→ *Capability-based security on Linux*

ARM platform support

Current focus on OMAP4 (Pandaboard)

- HDMI
- USB HID
- Networking
- SD-card

Multi-processor support

Kernels support SMP in different ways

transparent	Linux, Codezero
explicit API	L4ka::Pistachio, NOVA, Fiasco.OC

Challenge: Platform-independent API

→ *Similar problem to supporting real-time priorities*

ARM TrustZone

Promises solution for mobile security problems

- Two worlds: secure and non-secure world
- Run Genode in secure world
- Run Linux in non-secure world

→ *Genode bootstraps and supervises non-secure world*

→ *Genode implements security functions*

→ *Using base-hw platform*

A lot more...

- More light-weight device-driver environments
- IOMMU support on NOVA
- HelenOS Spartan kernel
- OSS
- Virtual NAT
- Genode on FGPA softcores
- Trusted computing
- Network of Genode systems
- New base platforms (Xen, Barrelfish, seL4)
- Language runtimes (D, Rust, Haskell, Go)

Thank you

What we covered today

Compositions

1. Virtualization techniques
2. Enslaving services
3. Dynamic workloads
4. Current ventures

What to do next...

Get involved

- Join the mailing list
- Check out the issue tracker
- Seek inspiration

  http://genode.org/about/challenges
- Discuss your ideas
- Start hacking!

More information and resources:

`http://genode.org`