Reaching puberty: How Genode is becoming a general-purpose OS

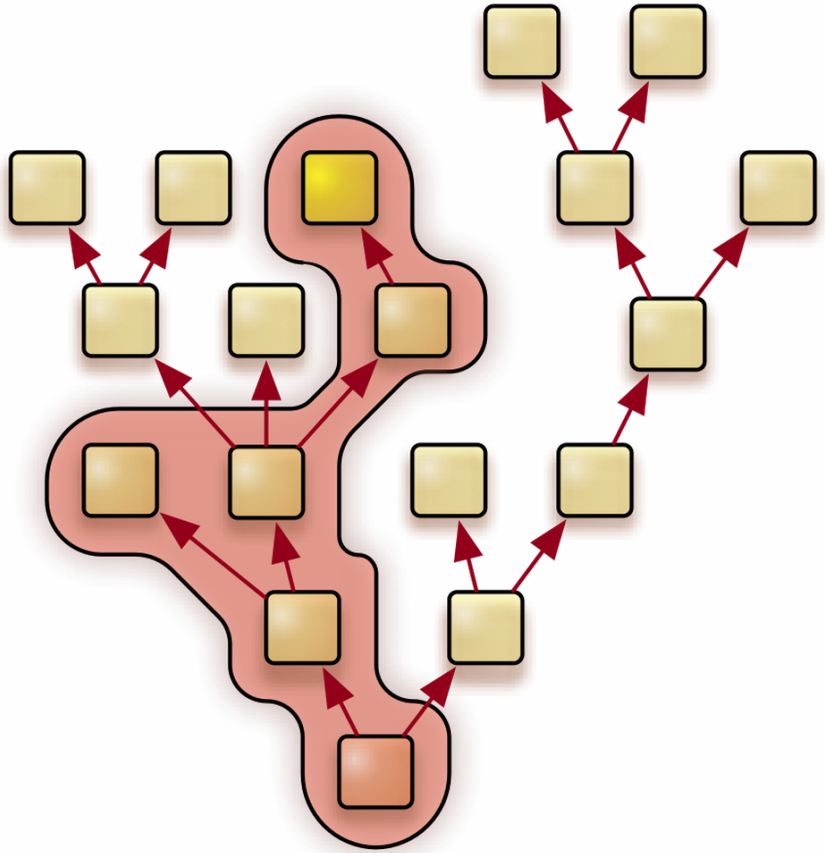
Norman Feske
<norman.feske@genode-labs.com>

Outline

1. Background
2. Noux runtime for Unix software
3. Challenges of dynamic system composition
4. Fundamental features
5. Current ventures

Outline

Genode in a nut shell



→ Application-specific TCB

Combined with virtualization

Genode OS Framework

FIASCO.OC

OKL4

CODEZERO

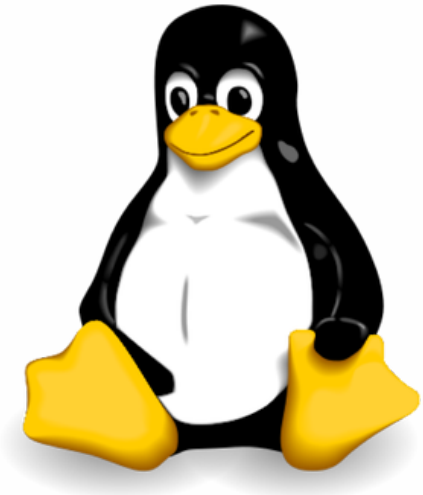
FIASCO

NOVA

Microhypervisor

L4Ka

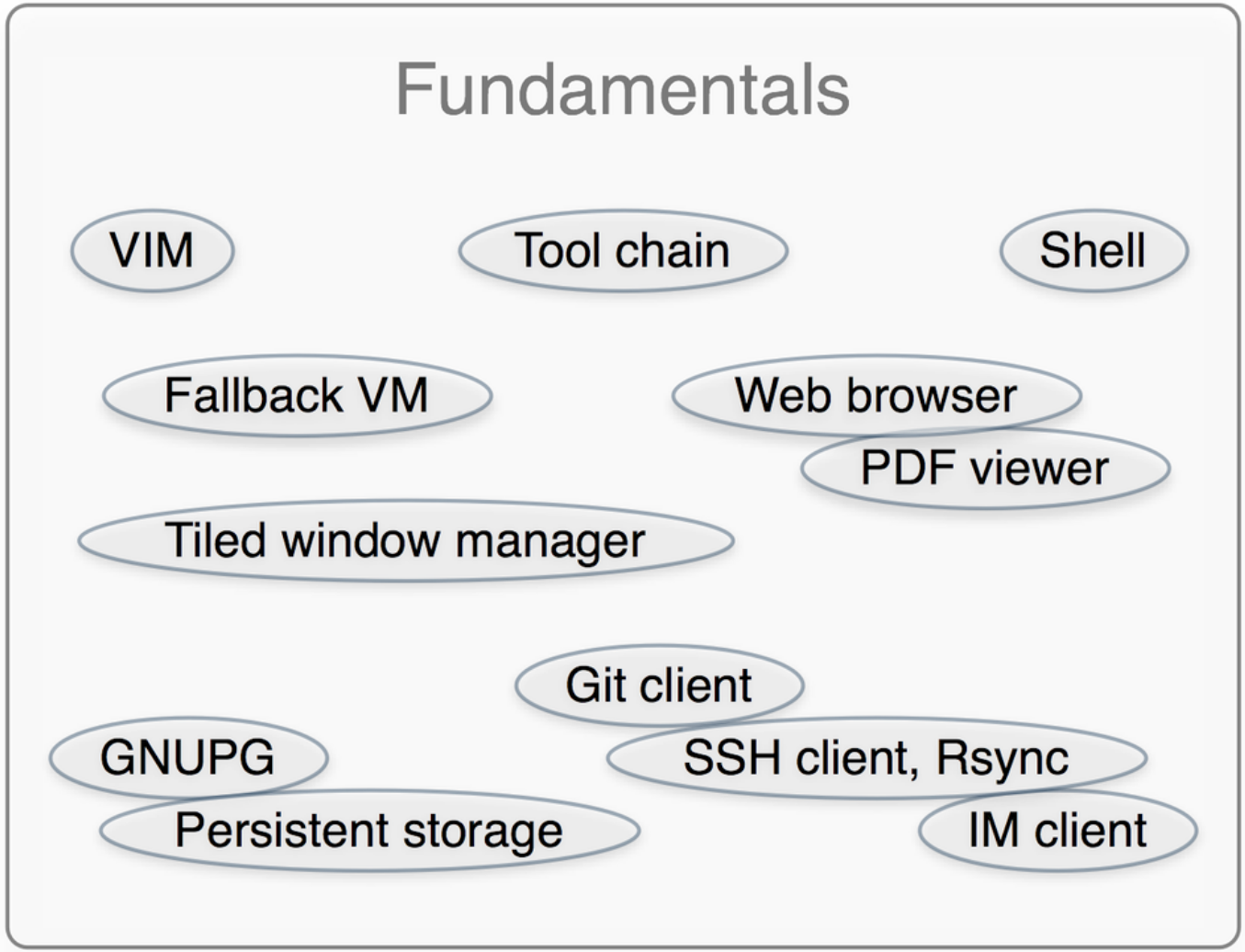
MicroBlaze



Genode OS Framework (2)

- 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

Eating our own dog food



Noux runtime for Unix software

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`

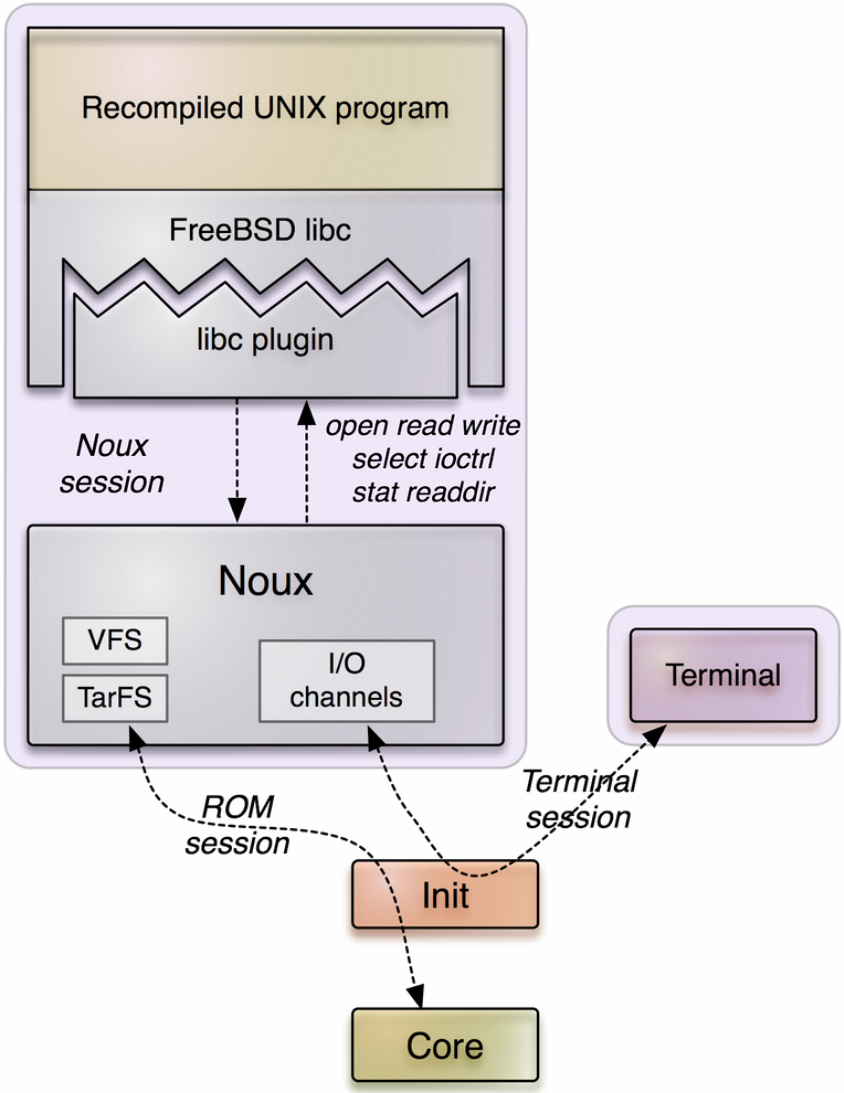
In contrast, Linux has more than 300 syscalls

Noux runtime for Unix software (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!

Noux runtime for Unix software (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...
- Supports stacked file systems
- Instance starts in fraction of a second
- Uses original GNU build system → Porting software is easy
- Two versions
  - `noux/minimal`
  - `noux/net` (includes TCP/IP)

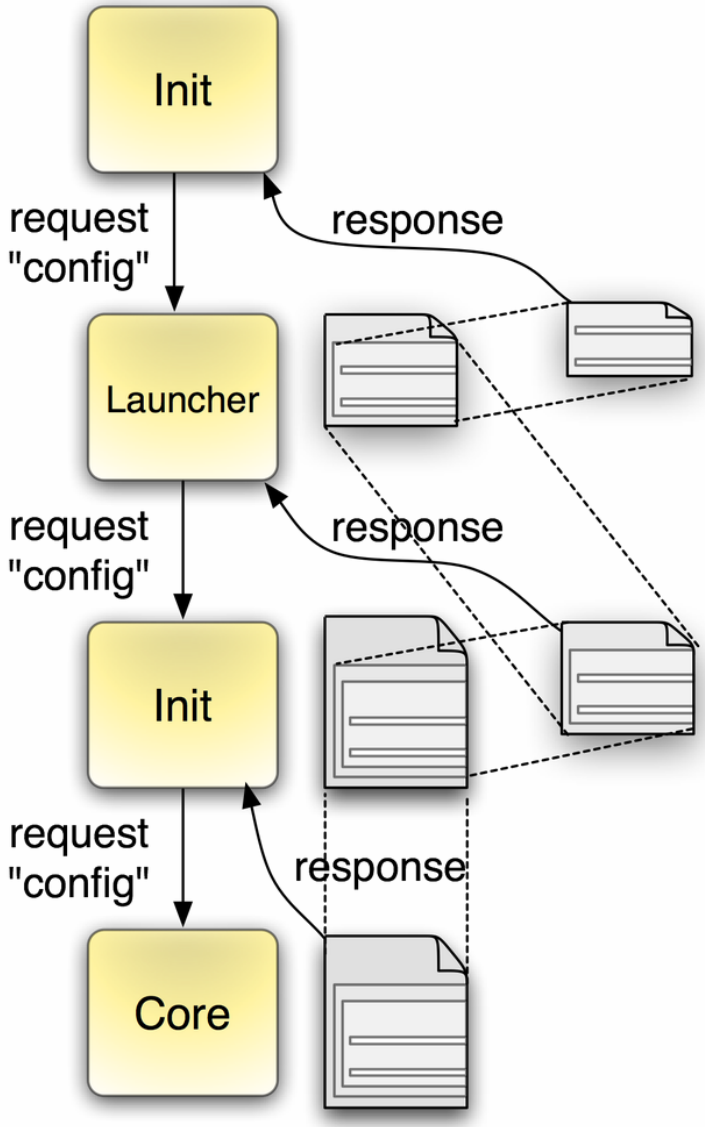
less than 5,000 LOC

Outline

Unified configuration concept

```
<config>
  <parent-provides> ... </parent-provides>
  <default-route> ... </default-route>
  ...
  <start name="nitpicker">
    ...
  </start>
  <start name="launchpad">
    ...
    <config>
      <launcher>
        <filename>init</filename>
        <config>
          <parent-provides> ... </parent-provides>
          <default-route>
            <any-service> <any-child/> <parent/> </any-service>
          </default-route>
          <start name="nit_fb">
            <resource name="RAM" quantum="6M"/>
            <config xpos="400" ypos="270" width="300" height="200" />
            <provides> <service name="Input"/>
                       <service name="Framebuffer"/> </provides>
          </start>
          <start name="l4linux">
            <resource name="RAM" quantum="1G"/>
            <config args="mem=52M l4x_rd=initrd.gz"/>
          </start>
        </config>
      </launcher>
    </config>
  </start>
</config>
```

Unified configuration concept (II)



Unified configuration concept (III)

→ Uniform syntax

→ Extensible through custom tags at each level

→ XML parser adds less than 300 LOC to TCB

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

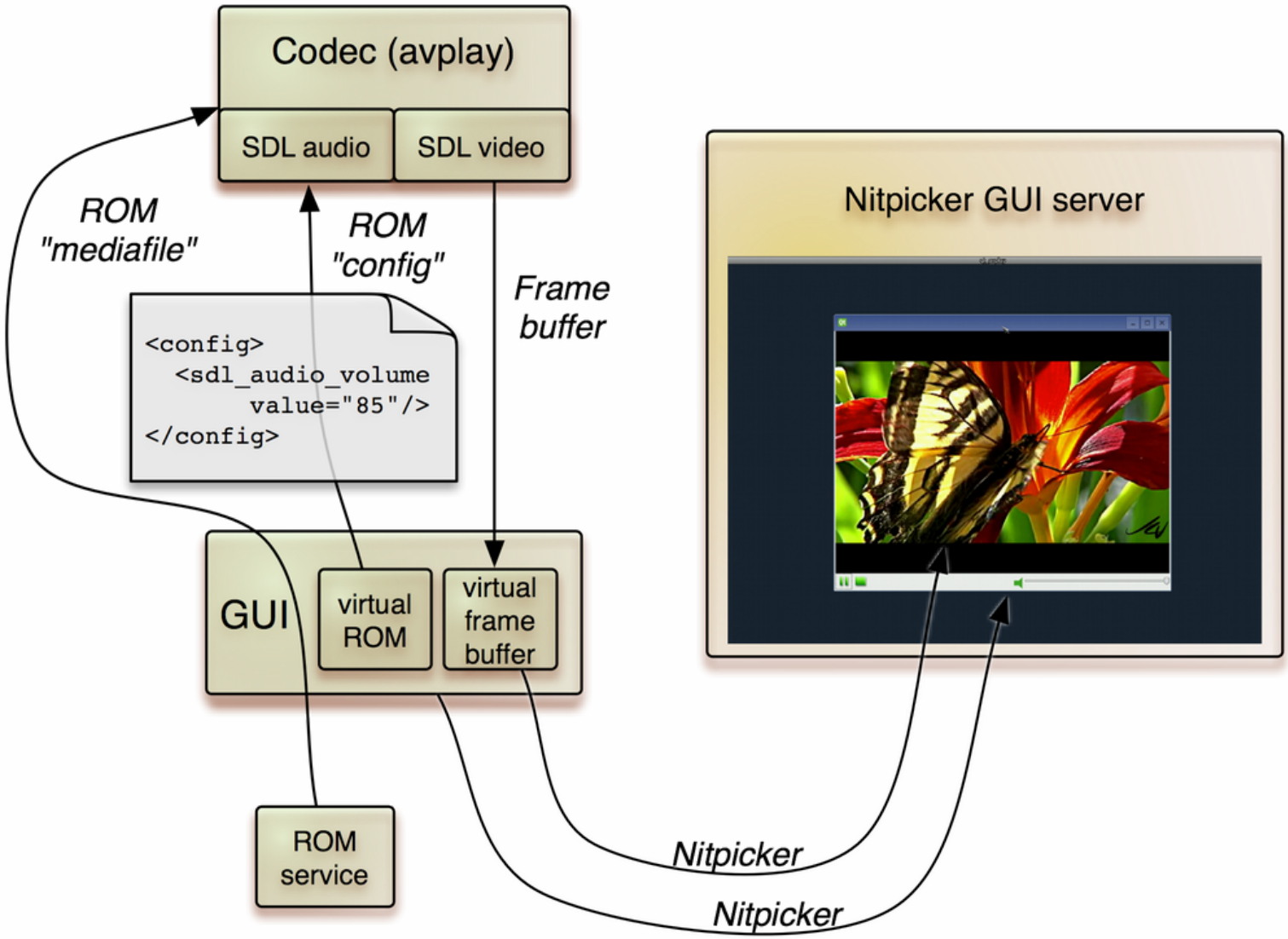
- Turn static config mechanism into dynamic mechanism

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)



Loader service

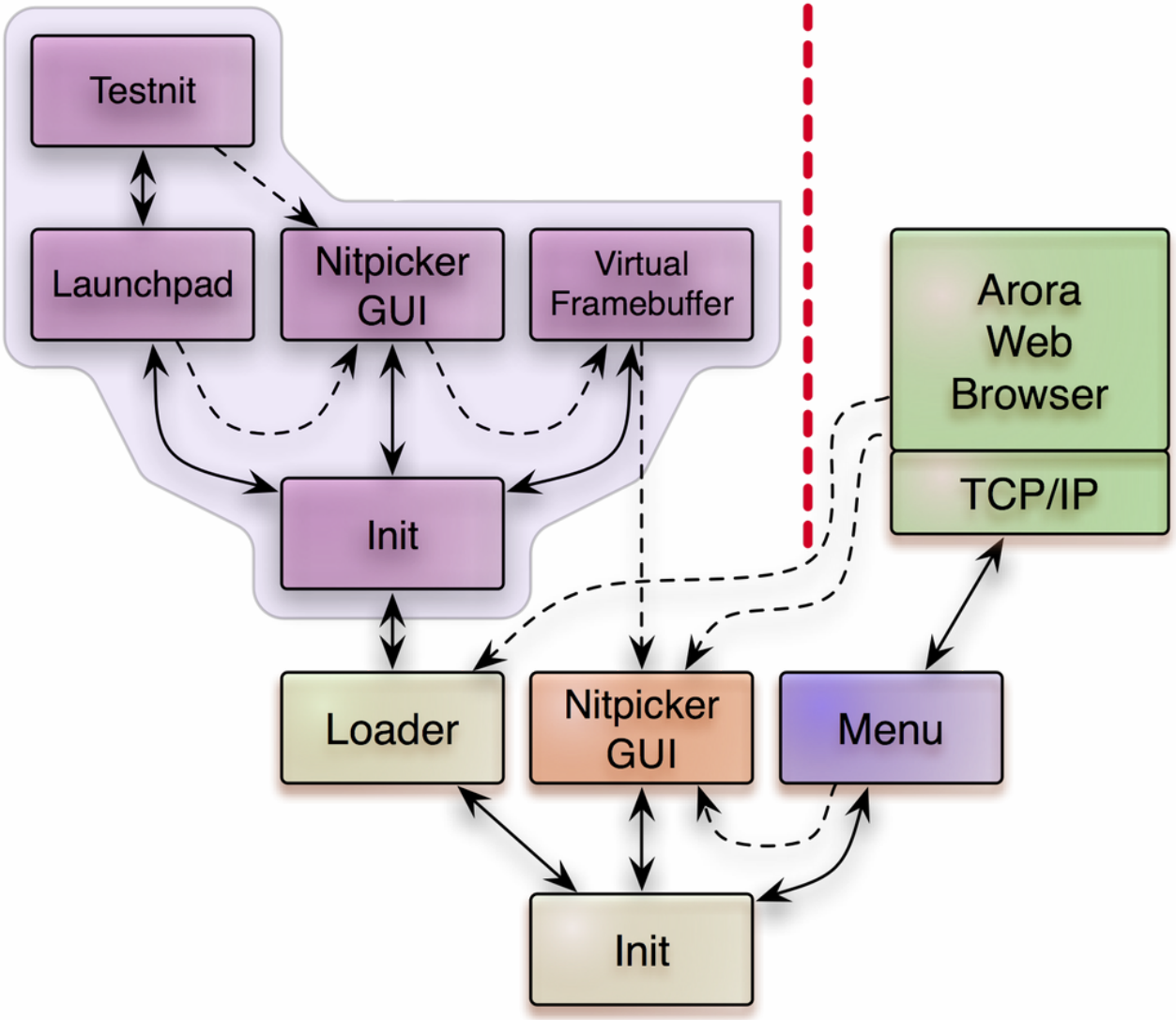
Challenges

- 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



Outline

File-system infrastructure

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`

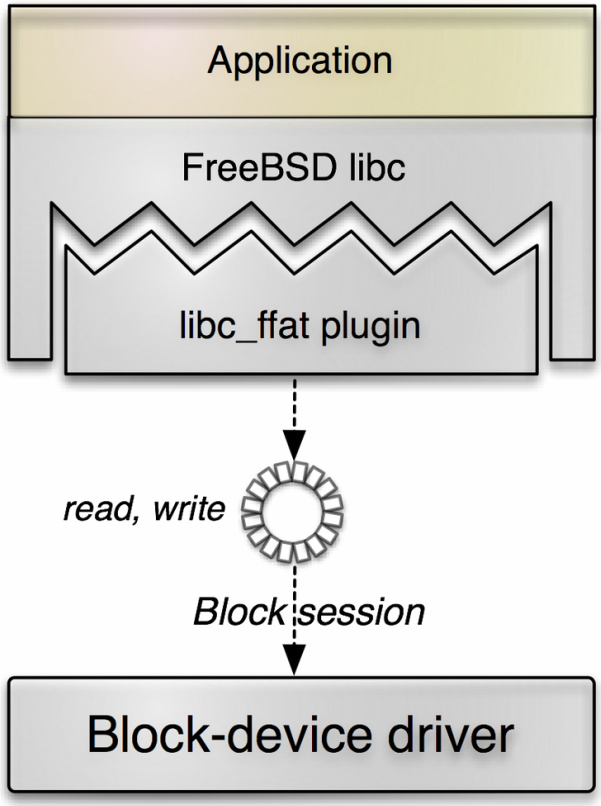
`libports/lib/mk/libc_lwip.mk`

`libports/lib/mk/libc_ffat.mk`

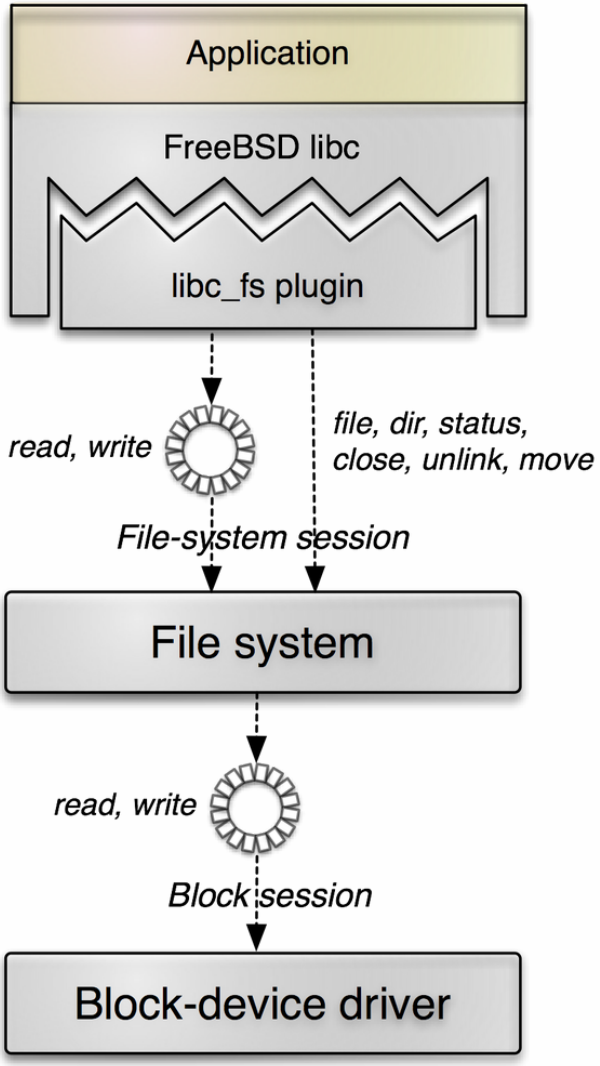
`libports/lib/mk/libc_lock_pipe.mk`

→ *application-specific plugins*

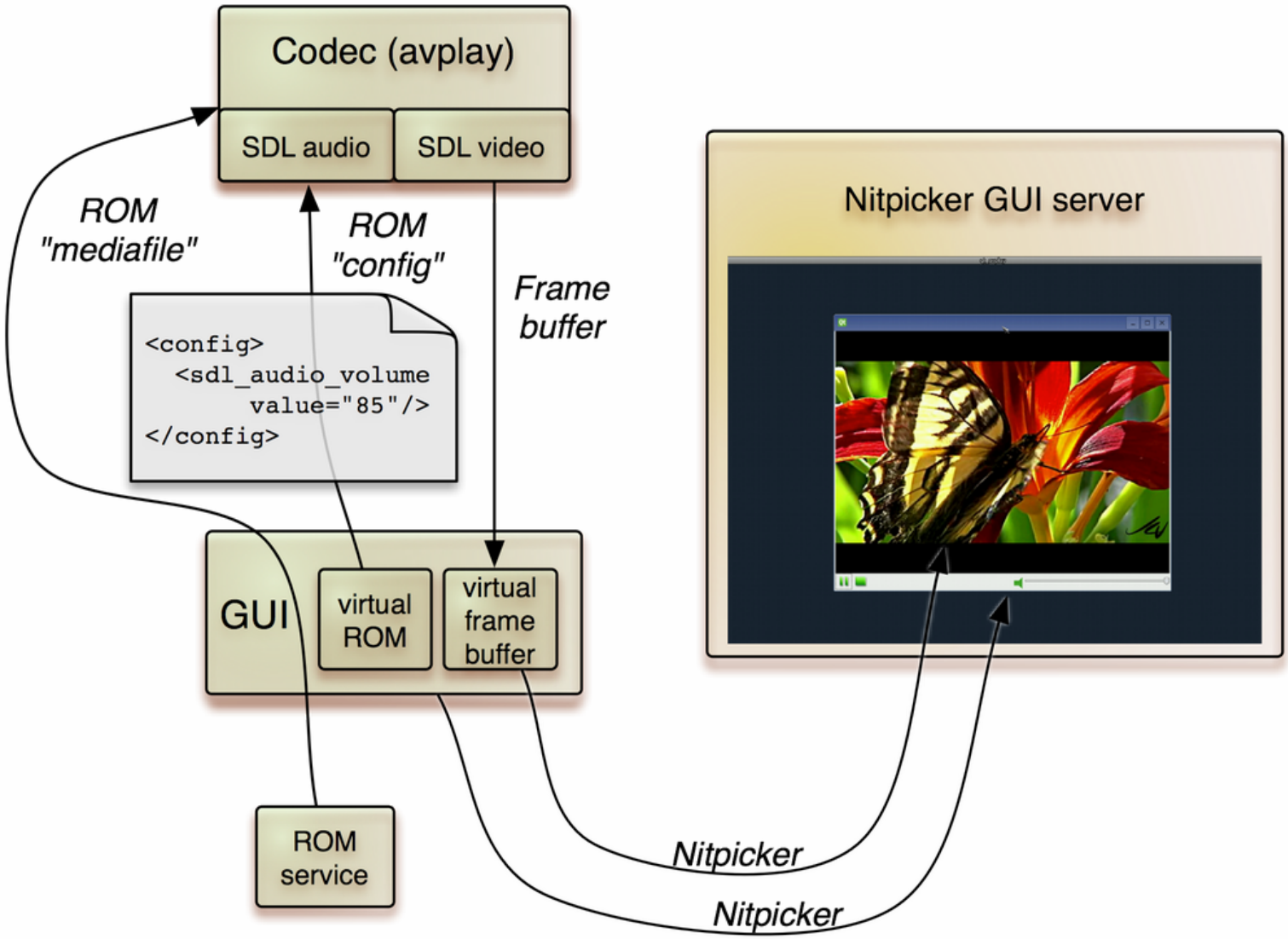
File-system infrastructure (2)



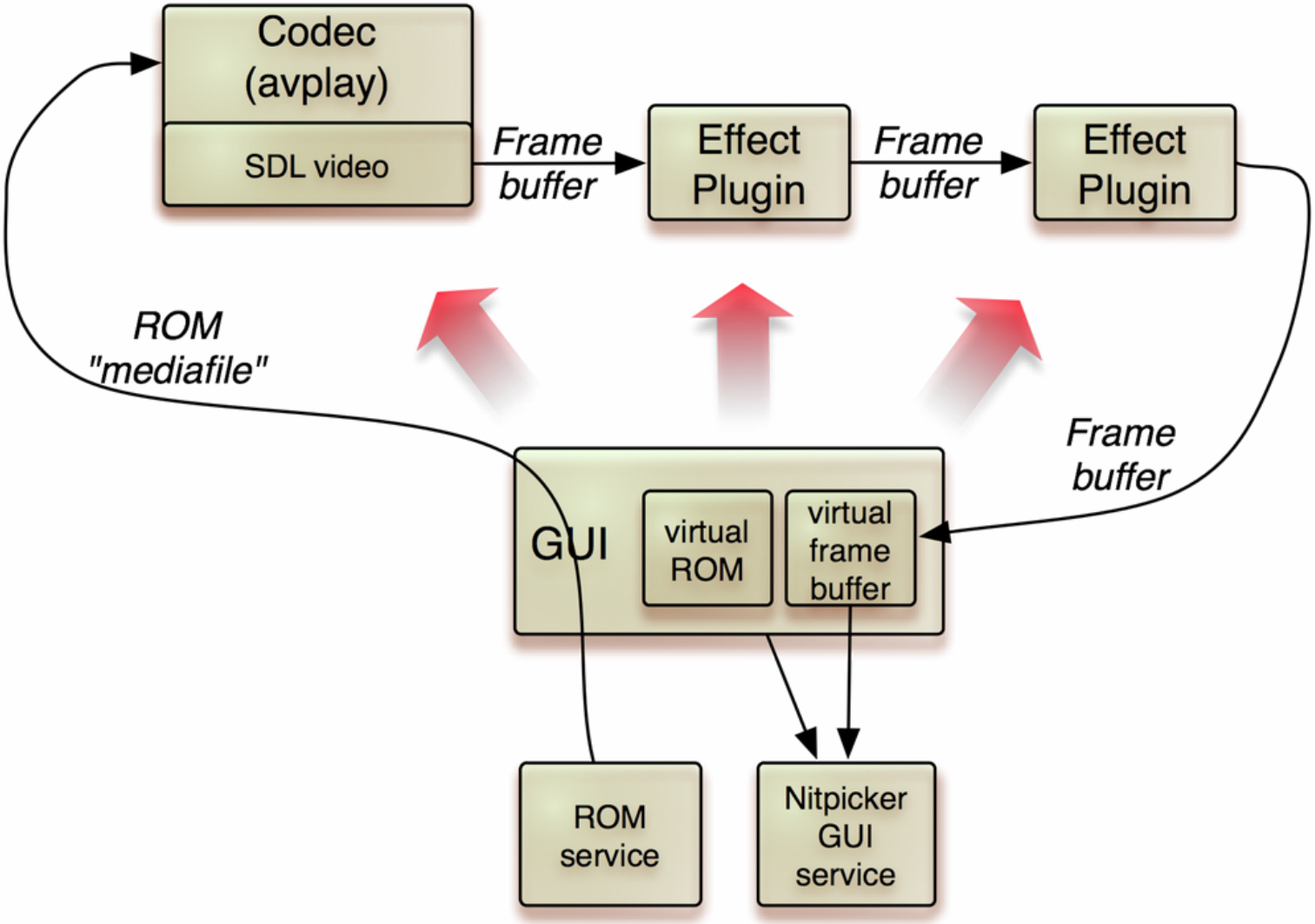
File-system infrastructure (3)



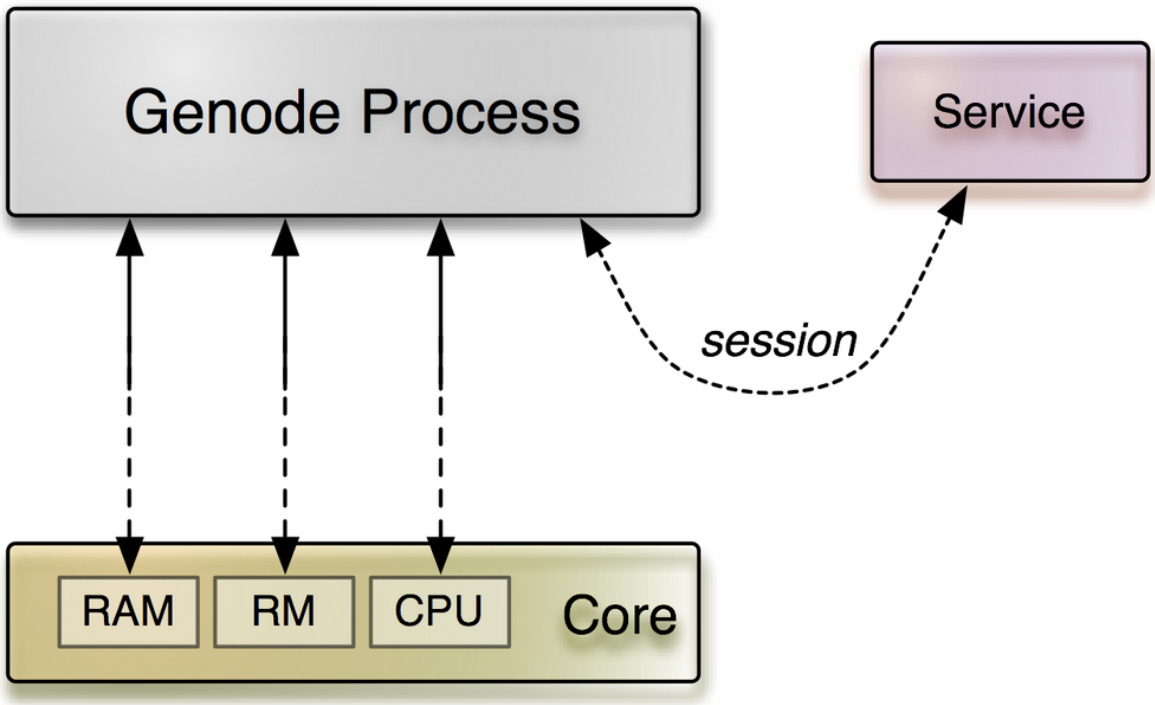
Media playback



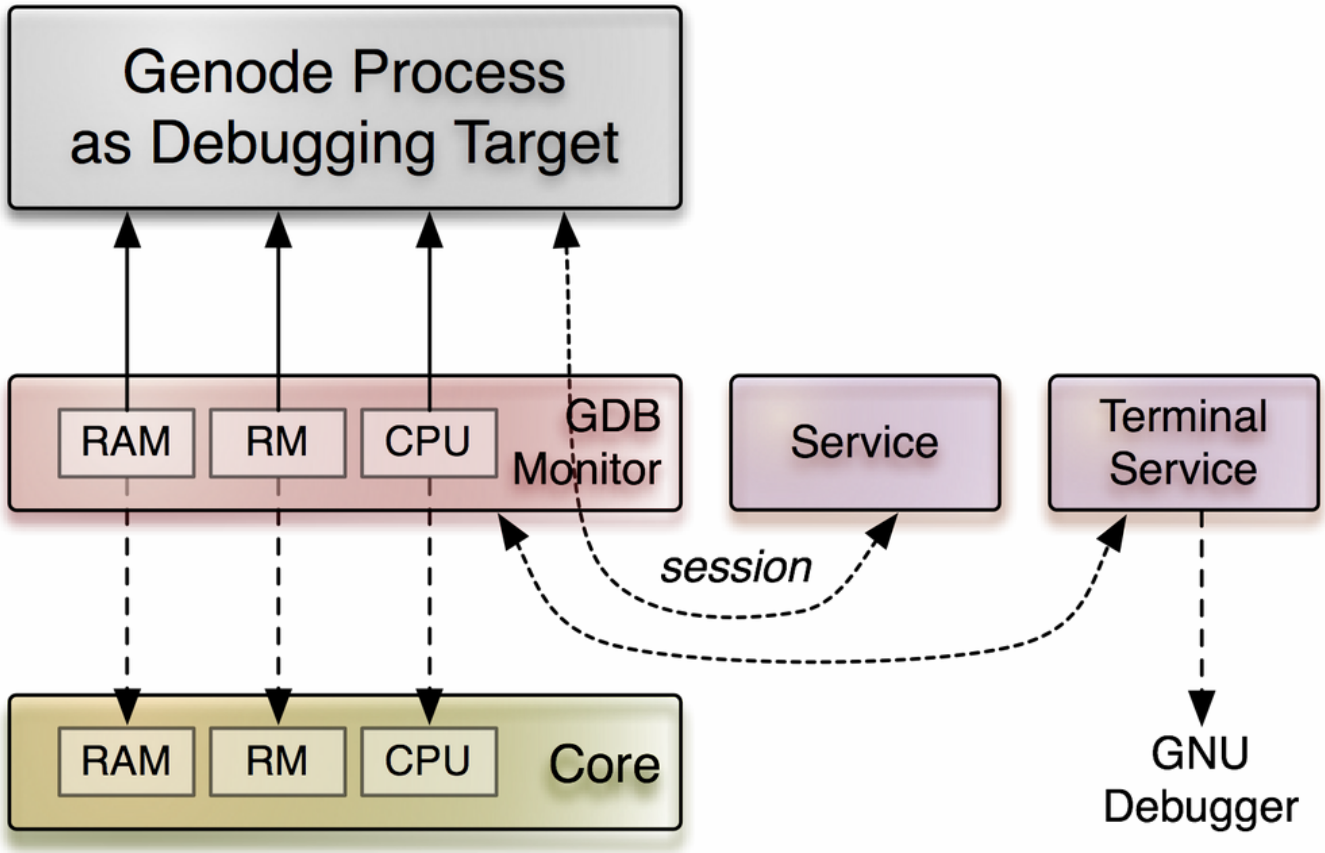
Media playback (2)



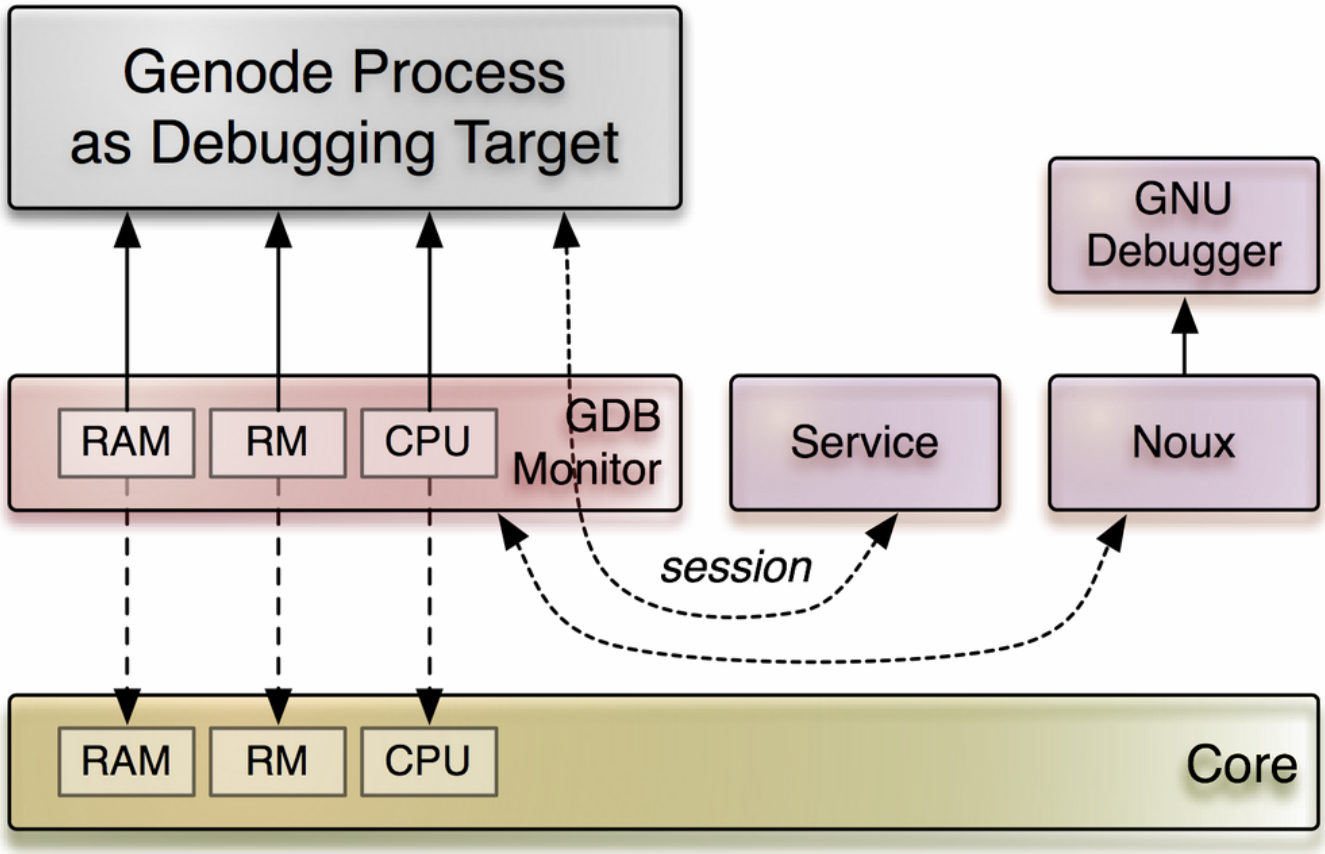
User-level debugging



User-level debugging (2)



User-level debugging (3)



Compiling Genode on Genode

Construction sites

- Kernels
- Base system
- C runtime, 3rd-party libraries
- Noux
- Porting the tool-chain components
  GCC, binutils, GNU make, findutils

→ Insightful lessons about application performance

Outline

User interface concept

Genode's architecture calls for tailored UI concept

Ingredients Nitpicker, framebuffer drivers, input drivers

Desired

- Convenient command-line interface
- Scripting
- Flexibility
  multi-head, virtual desktops, different window layouts
- Resource management

Performance and scalability

- Multi-processor support
  - NUMA
  - Challenge: Platform-independent API
  - Facilitating Genode's recursive structure
- Storage
  I/O scheduling, caching
- Networking (i. e., TCP/IP performance)
- Tools
  Profiling, debugging, tracing

Networking and security

- IOMMU support on NOVA
- Trusted computing

  $\rightarrow$ Network of Genode systems
- Capability-based security on Linux

Noux: Unix networking tools

Needed command-line tools

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

Approach

Integrate lwIP into Noux runtime

→ *One TCP/IP stack per Noux instances*

A lot more...

- More light-weight device-driver environments (e. g., OSS)
- ARM TrustZone
- Hardware support (e. g., ARM SoCs)
- HelenOS Spartan kernel
- “Real” file system
- Virtual NAT
- Genode on FPGA softcores

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