Introducing a radically componentized GUI architecture

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

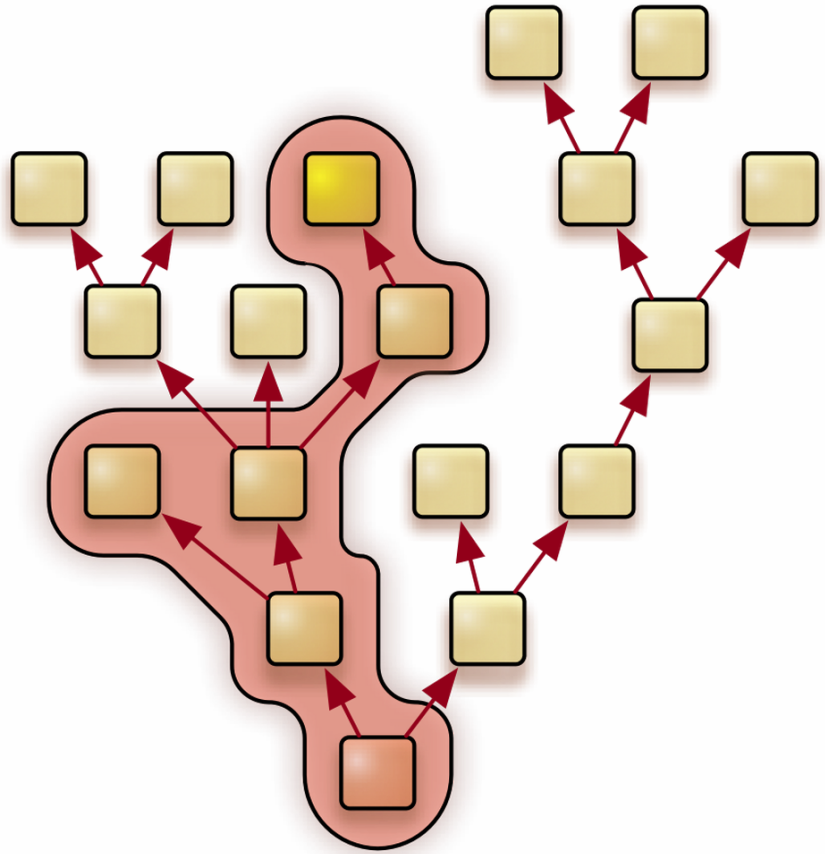
Outline

1. Starting point
2. Ingredients
3. Challenges and solutions
4. Next steps

Outline

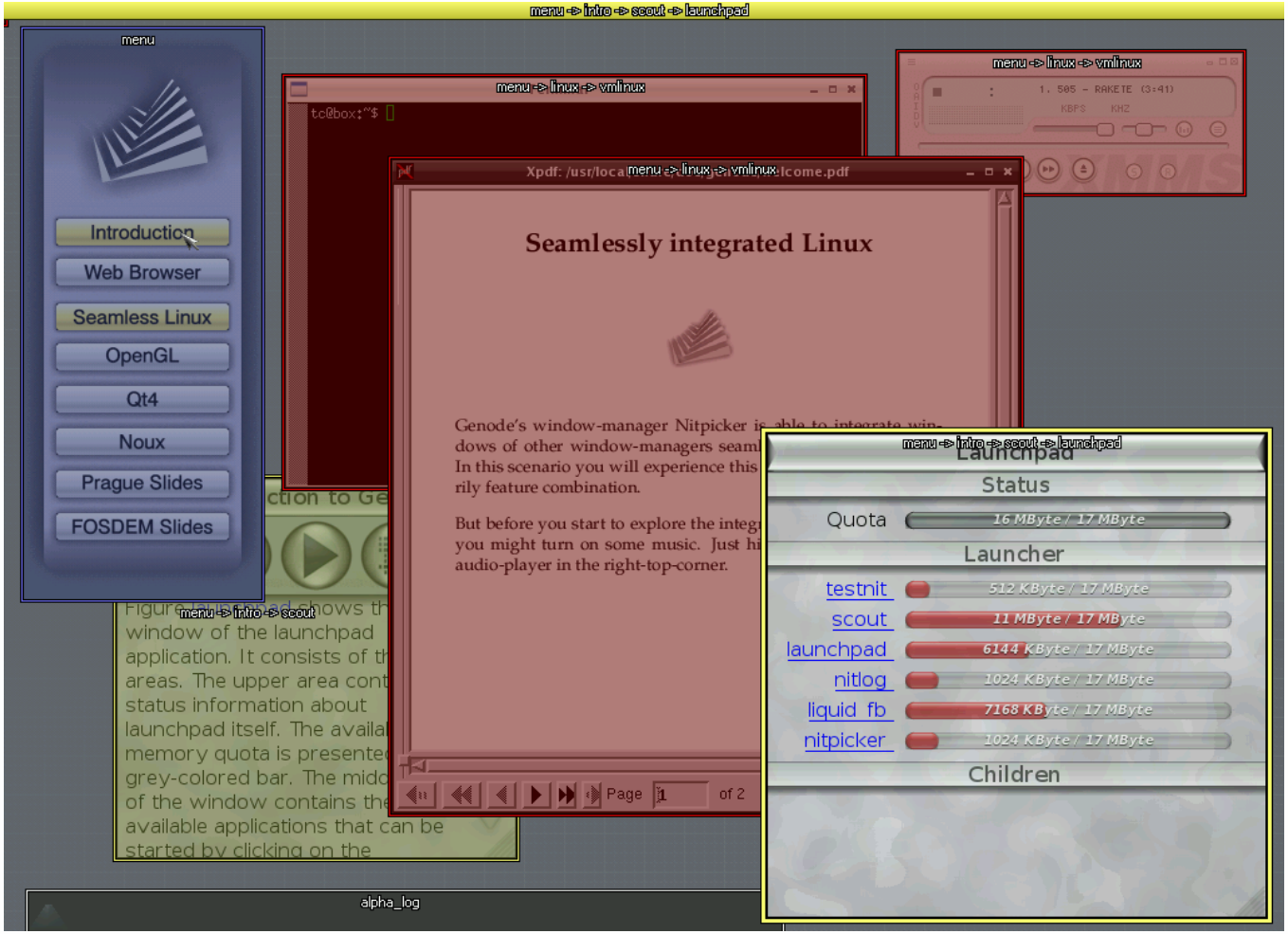
1. Starting point
2. Ingredients
3. Challenges and solutions
4. Next steps

Starting point - Genode

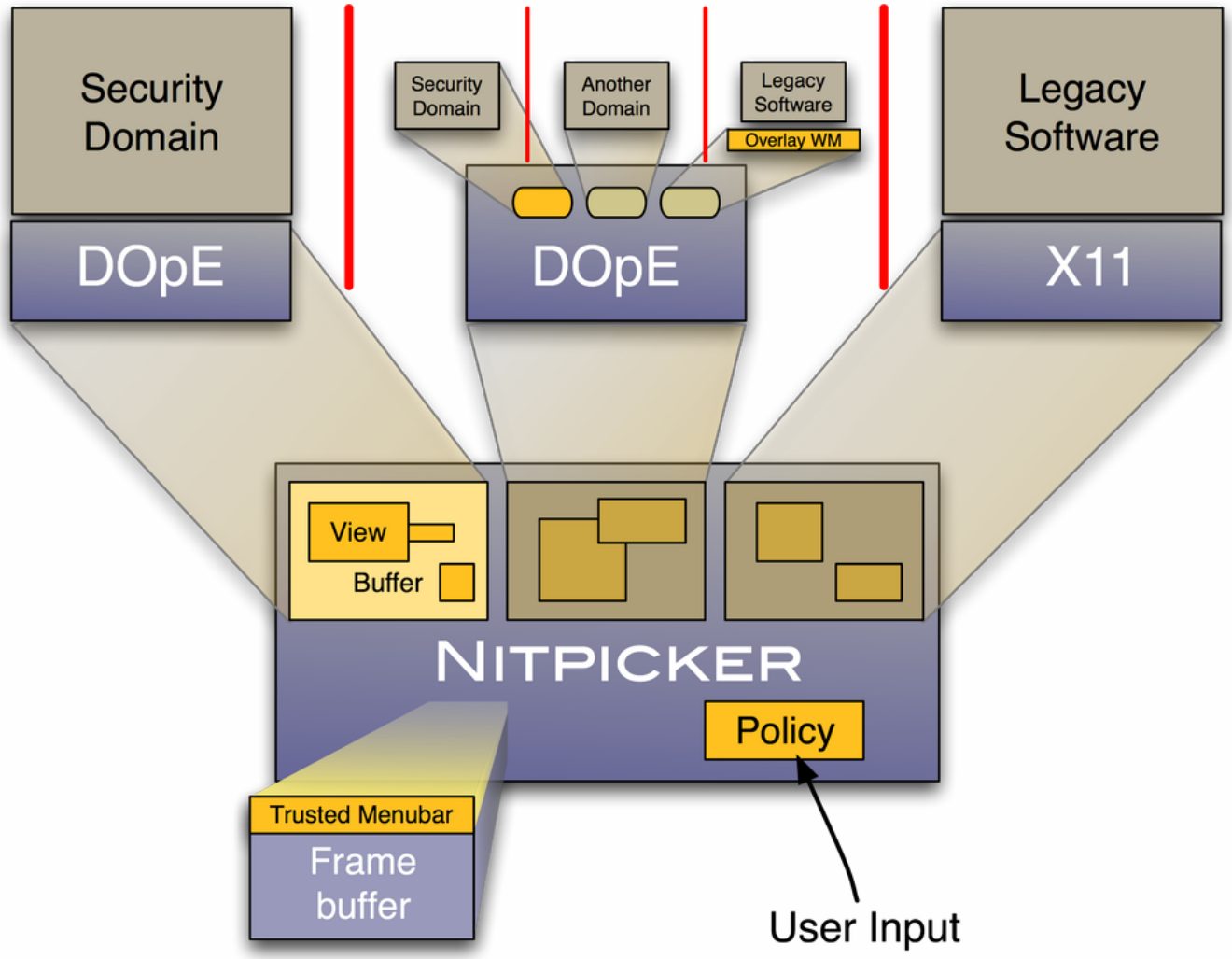


→ Application-specific TCB

Starting point - Nitpicker



Starting point - Nitpicker



Starting point

Starting point

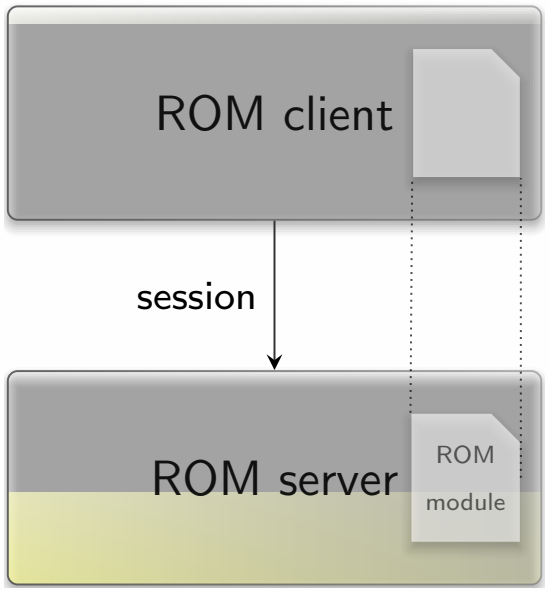
- Low-complexity GUI server (nitpicker)
- Toolkits
  - Qt5
  - DOpE
  - Custom widget set
- Hard-wired policy

Goal $\rightarrow$ Desktop environment

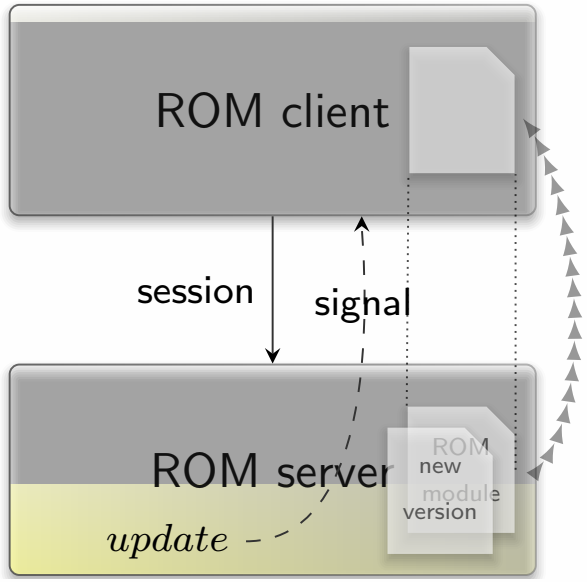
- Retain low TCB complexity
- Accommodate a great variety of use cases

Outline

ROM session interface

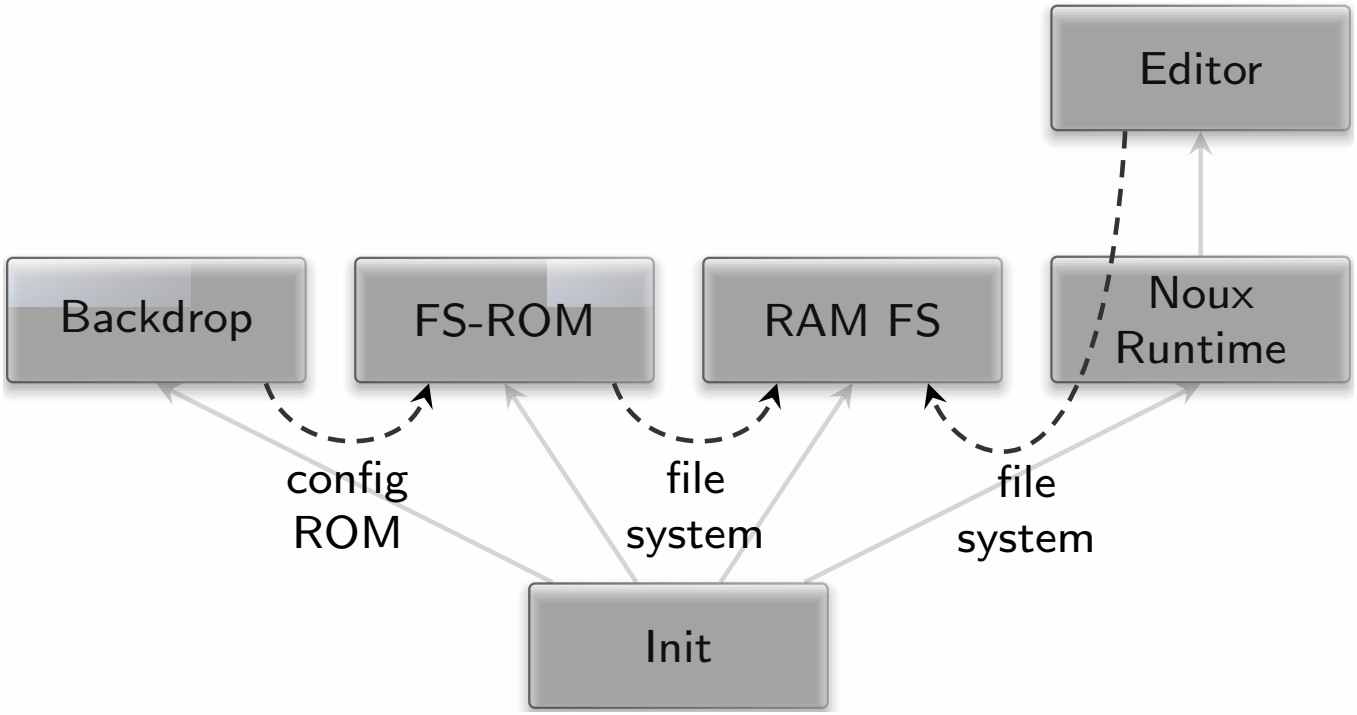


ROM session interface (2)



Transactional update of a ROM session

ROM session interface (3)



ROM session interface (4)

Demo

Report session interface

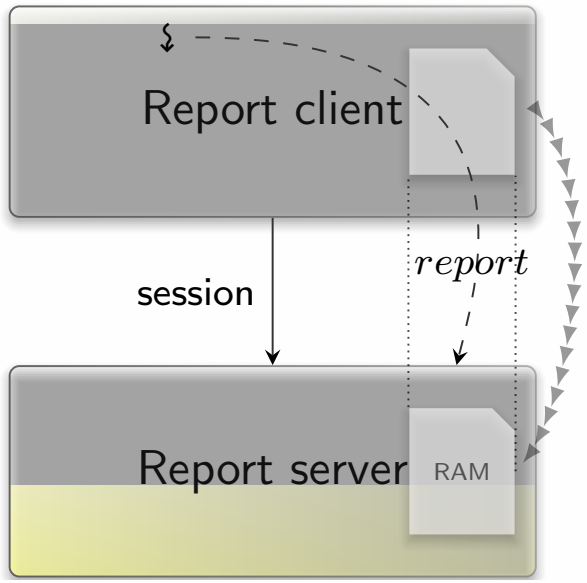
Existing mechanisms for propagating information

- Configuration defined at startup
- Policy defined at session-creation time
- Session interfaces
- Dynamic configuration changes

What is needed in addition?

- Components need to publish internal state, e. g.,
  - Driver: Report available device resources
  - Component: Report feature set
  - Applications: User notifications
  - Propagating error conditions

Report session interface (2)



Publisher-subscriber mechanism

Combining “Report” and “ROM” session interfaces

- The report_rom server provides
  - “Report” service
  - “ROM” service
- Stores reports using report-session labels as keys
- Controls access using ROM-session labels as selectors
- Triggers ROM-changed signals on incoming reports

→ **Generic publisher-subscriber mechansism**

- Composeable with existing ROM-using components
- Can be instantiated many times

Outline

Flexibility of Nitpicker

Nitpicker's built-in policy stands in the way

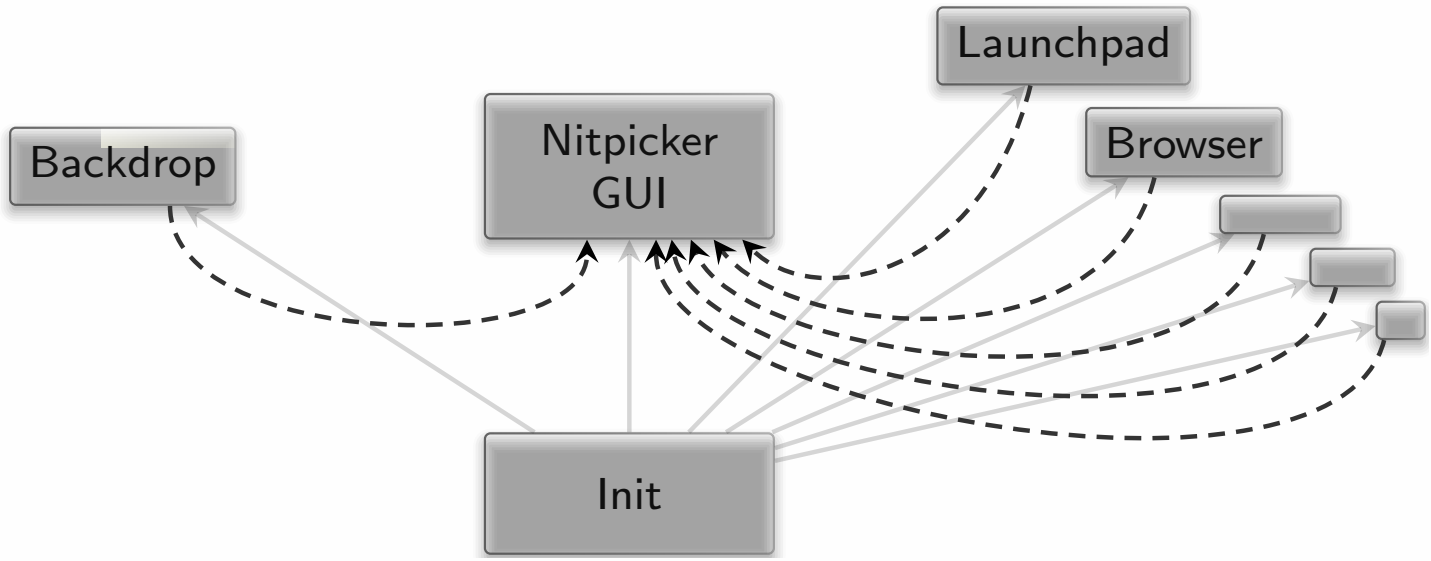
New configuration concept

- Domains
- Layering

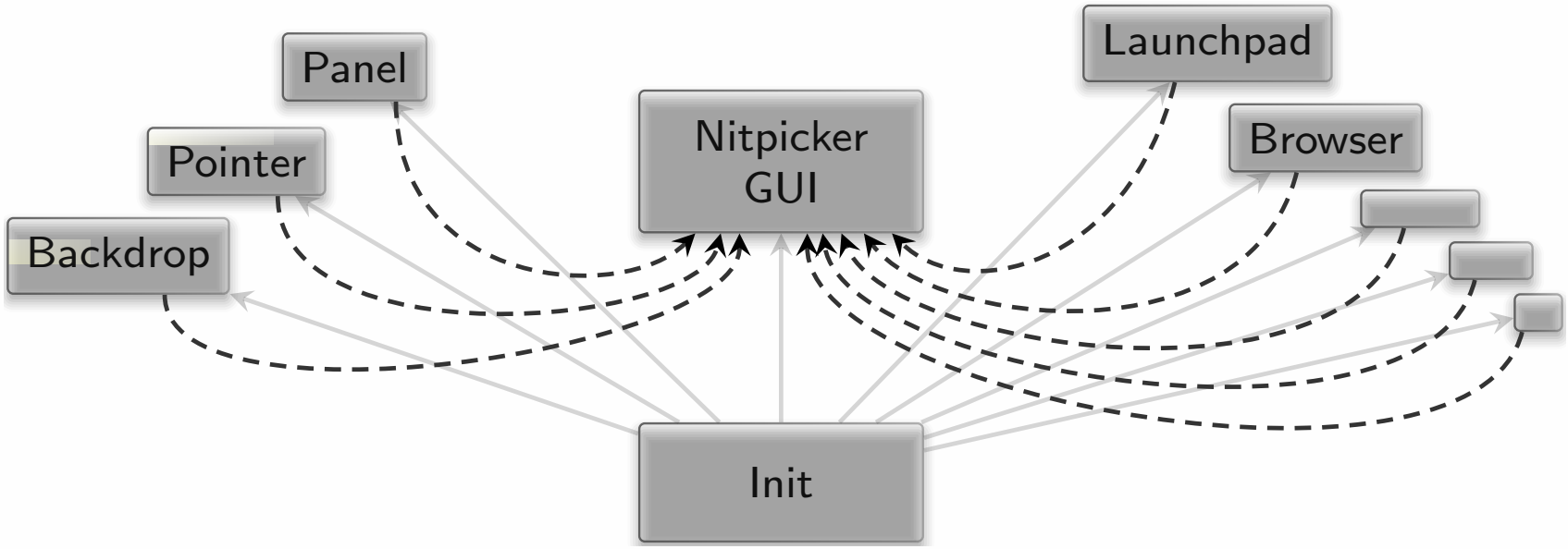
→ **Separation of policy from the nitpicker server**

- Pointer
- Status bar

Nitpicker with built-in policies



Policy as external components



Domains example

Demo

Transitions

How to smoothly toggle the visibility of the windows?

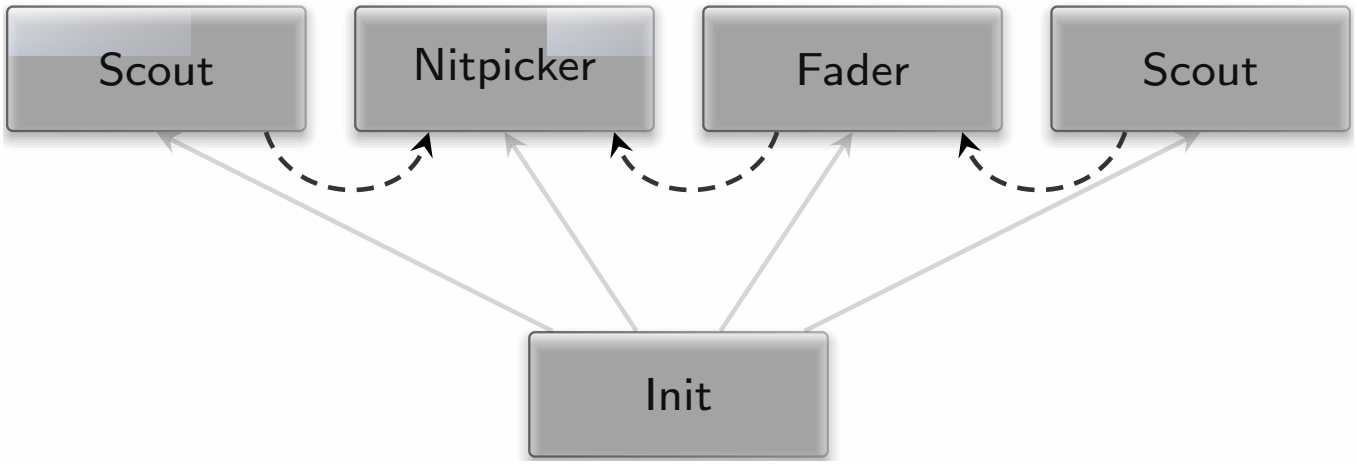
- Adding fading feature to the application?
  → Increase application complexity
  → Modifications needed per application

- Adding fading feature to nitpicker?
  → Increase complexity of nitpicker

Solution

→ Move fading feature to separate component

Transitions (2)



Transitions (2)

Demo

Launcher

Starting point

- Demo menu (monolithic application)
- Based on pre-rendered PNG images
- Customization is labour intensive

Customizable launcher

- Runtime-generated widgets
  $\rightarrow$ *complex (e. g., relies on libc, libpng, zlib)*

Launcher (2)

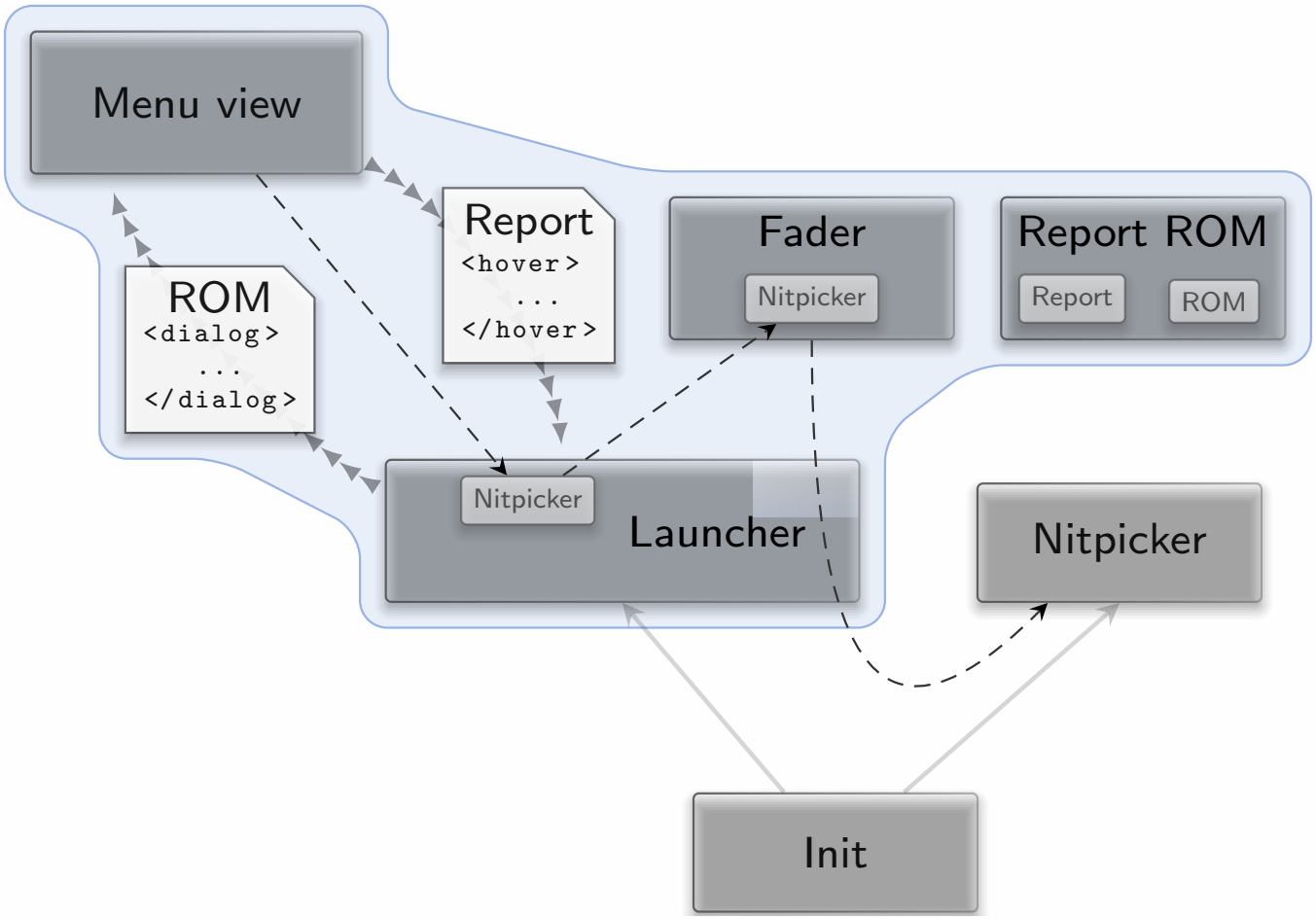
How to keep the complexity of the launcher low?

- Launcher is parent of all started subsystems
  $\rightarrow$ belongs to the trusted computing base
- Appealing presentation comes with complexity

Solution

1. Turn launcher into a multi-component application
2. Sandboxed widget-rendering component

Launcher (3)


Menu view
Report
<hover>
...
</hover>
Fader
Nitpicker
Report ROM
Report
ROM
ROM
<dialog>
...
</dialog>
Nitpicker
Launcher
Nitpicker
Init

Launcher (4)

Demo

Window management

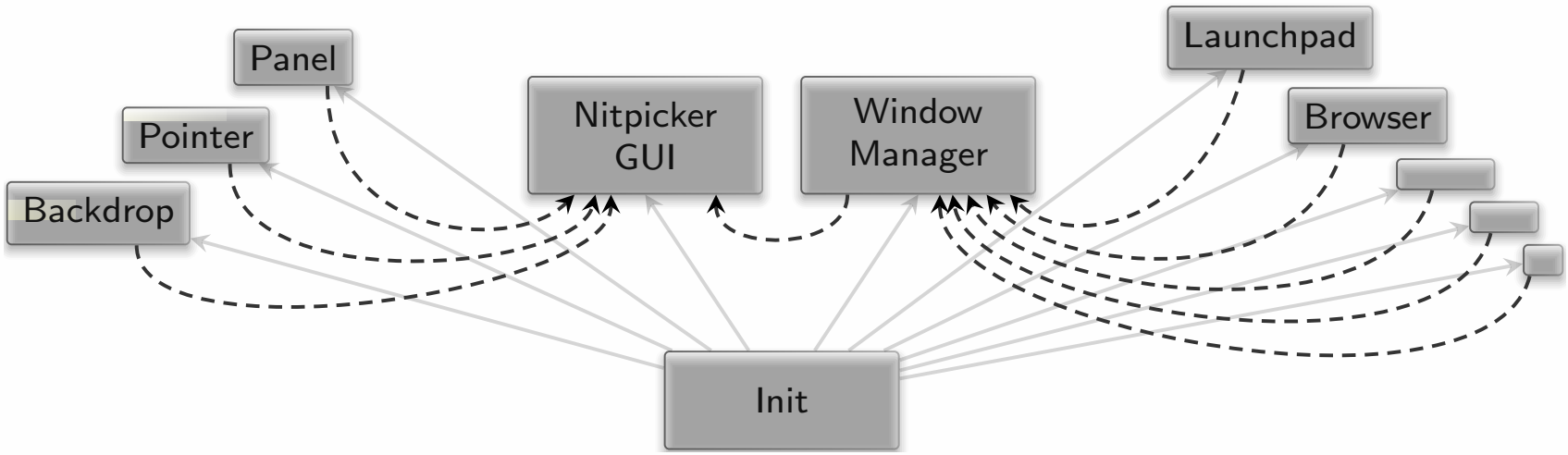
Starting point

- Genode lacked a coherent window manager
- Application-specific window management

Problem

- Diversity of tastes and expectations by users
- There is no a single solution for everyone

Window management (2)

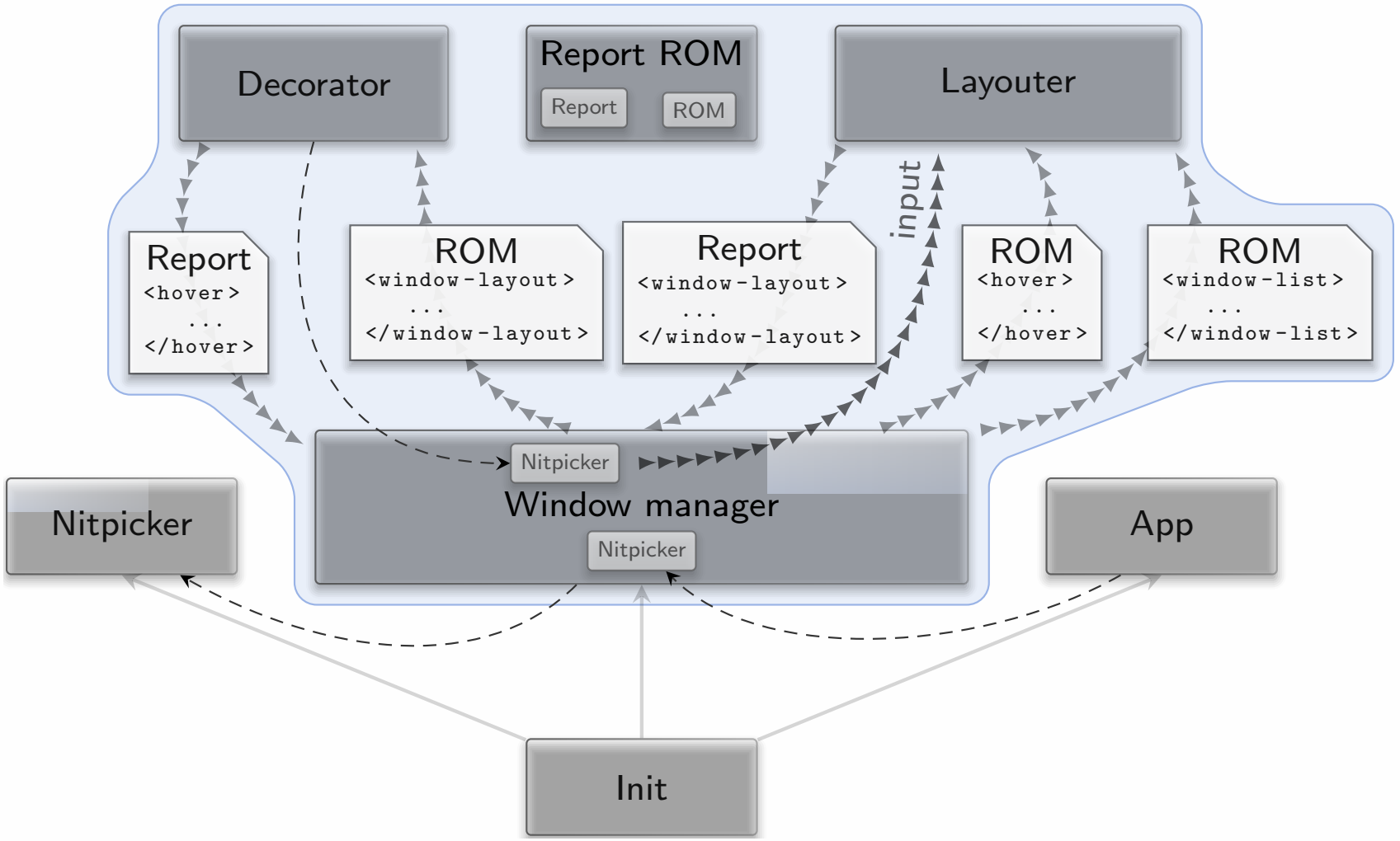


Window management (3)

De-componentized window manager

- Provides “Nitpicker” interface (compatibility)
- Layouter (defines behavior)
- Decorator (defines look)
- Layouter and decorator are sandboxed

Window management (4)



Decorator

Demo

TCB complexity of window management

TCB footprint of the window manager

- No libc dependency
- Adds less than 3,500 SLOC

Further TCB reduction

- Multiple window-manager instances
- Each instance assigned to a different nitpicker domain

Screen resolutions

How to support different screen resolutions?

- The screen resolution used to be hard-wired at build time
  - VESA driver configuration
  - Background image of the matching size

Solution

1. Detection heuristics in the VESA driver
2. Resolution-independent backdrop
3. Dynamic framebuffer mode updates

Screen resolutions

Demo

Outline

Next steps

- Alternative window layouters and decorators
- Capability-based desktop environment
- Using Genode for day-to-day computing

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`