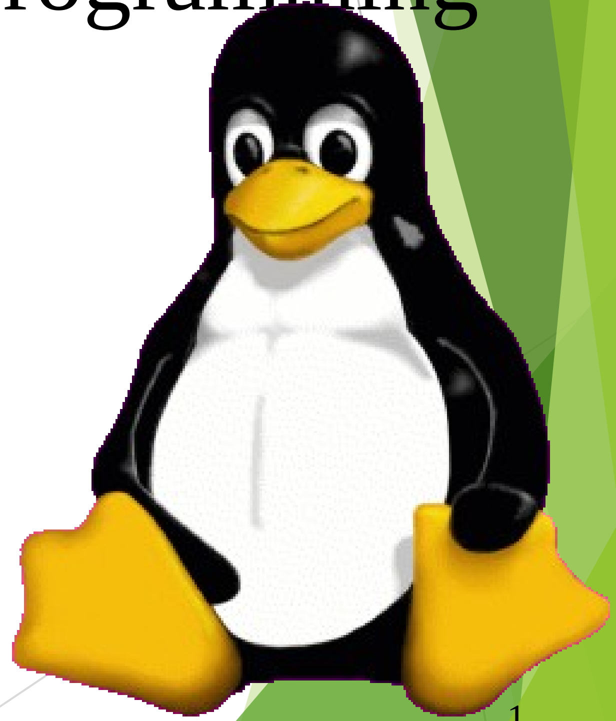
Advanced C and System Programming

Anandkumar



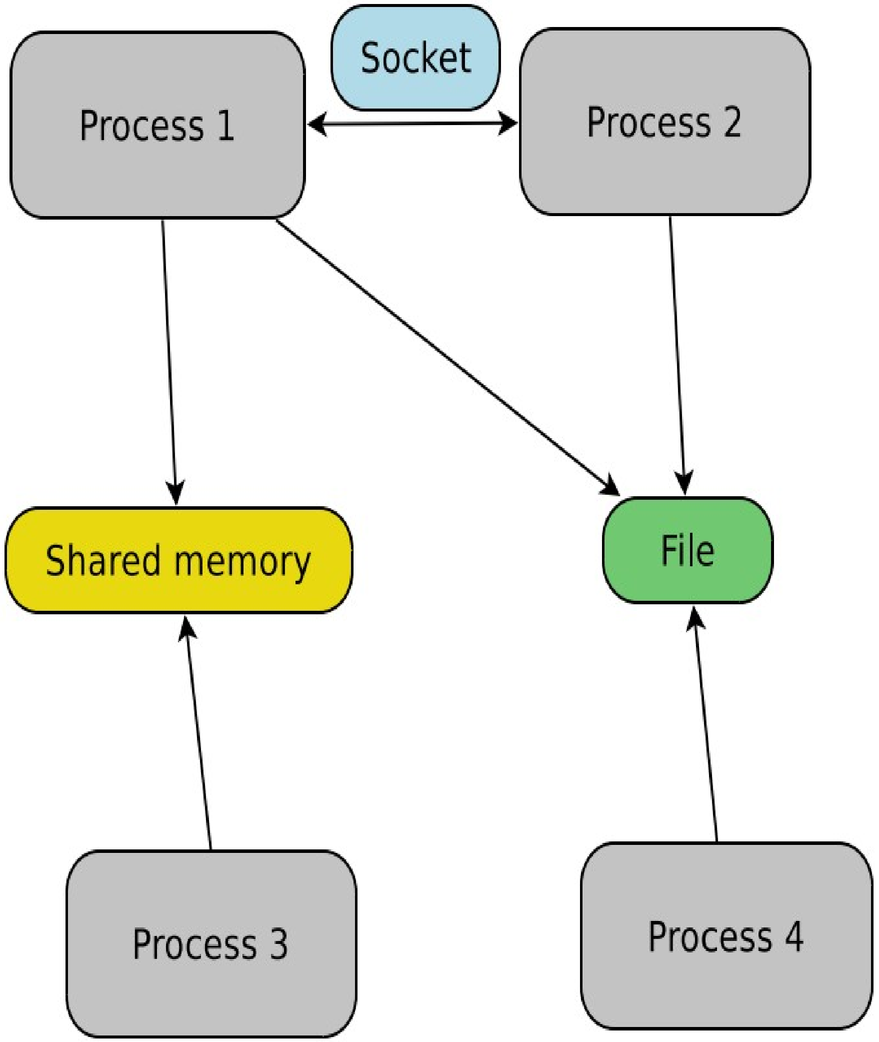
D-Bus

D-Bus

- Created in 2002
- Is part of the *freedesktop.org* project
- Maintained by RedHat and the community
- Is an Inter-process communication mechanism
- Initiated to standardize services of Linux desktop environments

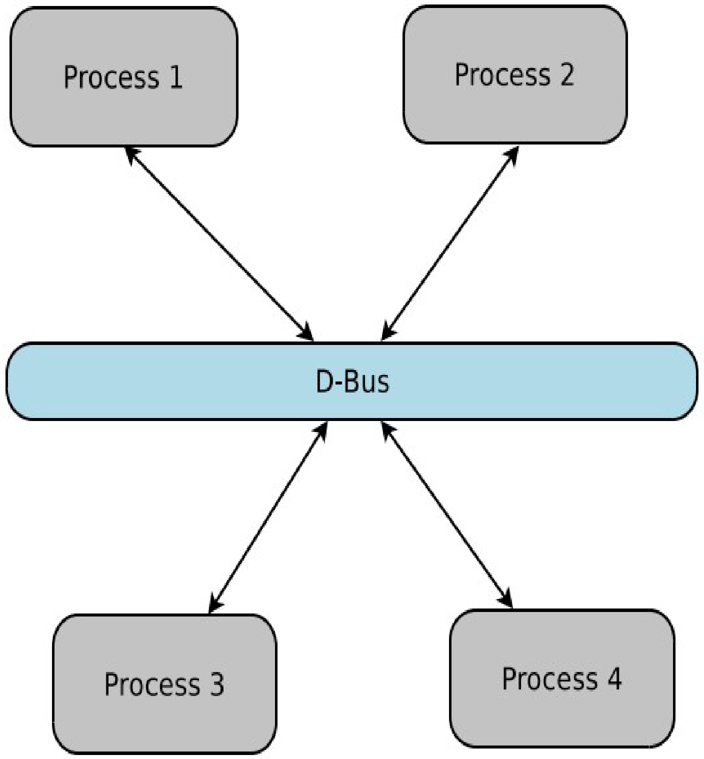
D-Bus

- Mechanisms allowing processes to communicate with each other
  - **Shared memory**: read/write into a defined memory location
  - **Memory-mapped file**: same as `shared memory` but uses a file
  - **Pipe**: two-way data stream (standard input / output)
  - **Named pipe**: same as `pipe` but uses a file (FIFO)
  - **Socket**: communication even on distant machines
  - and others



D-Bus

- Uses the socket mechanism
- Provides software bus abstraction
- Way simpler than most alternatives



D-Bus

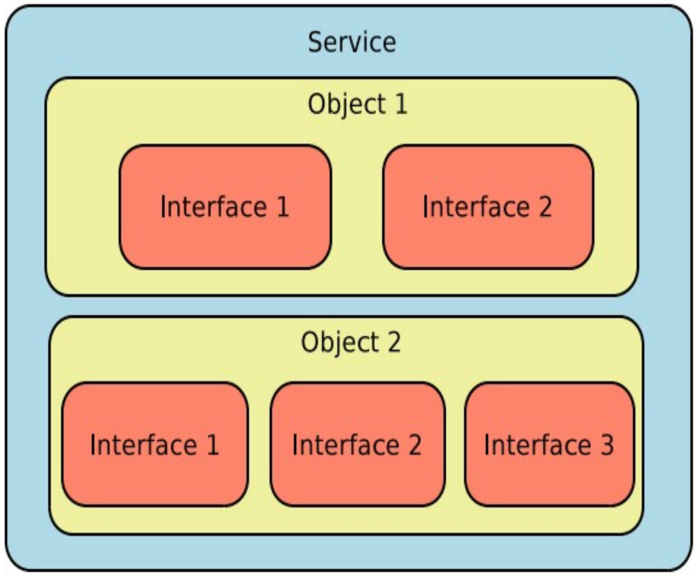
- D-Bus includes:
  - `libdbus`: a low-level library
  - `dbus-daemon`: a daemon based on libdbus. Handles and controls data transfers between DBus peers
  - two types of busses: a `system` and a `session` one. Each bus instance is managed by a `dbus-daemon`
  - a security mechanism using `policy` files

D-Bus

- System bus
  - On desktop, a single bus for all users
  - Dedicated to system services
  - Is about low-level events such as connection to a network, USB devices, etc
  - On embedded Linux systems, this bus is often the only D-Bus type
- Session bus
  - One instance per user session
  - Provides desktop services to user applications
  - Linked to the X session

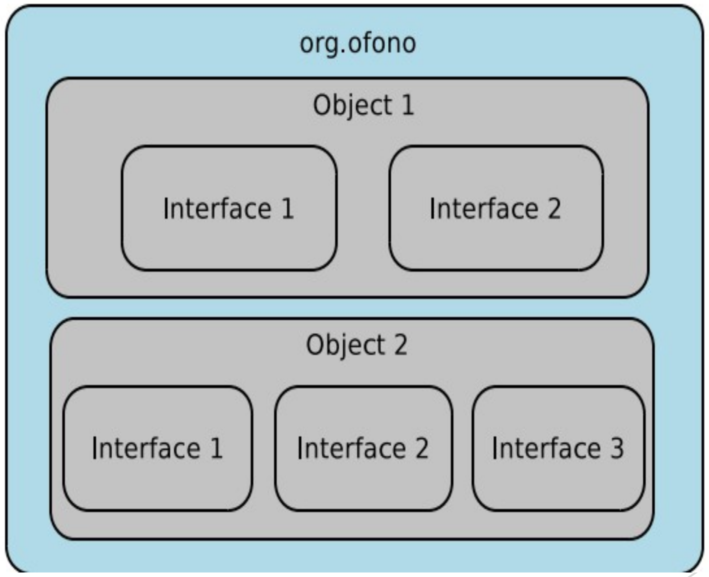
D-Bus

- D-Bus is working with different elements:
  - Services
  - Objects
  - Interfaces
  - Clients: applications using a D-Bus service
- One D-Bus *service* contains *object(s)* which implements *interface(s)*



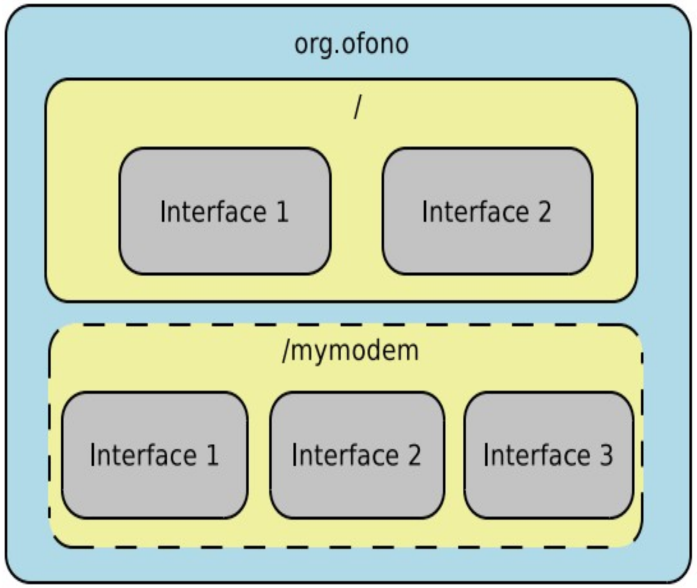
D-Bus Service

- An application can expose its services to all D-Bus users by registering to a bus instance
- A `service` is a collection of `objects` providing a specific set of features
- When an application opens a connection to a bus instance, it is assigned a unique name (ie `:1.40`)
- Can request a more human-readable service name: the **well-known name** (ie `org.ofono`) See the freedesktop.org specification



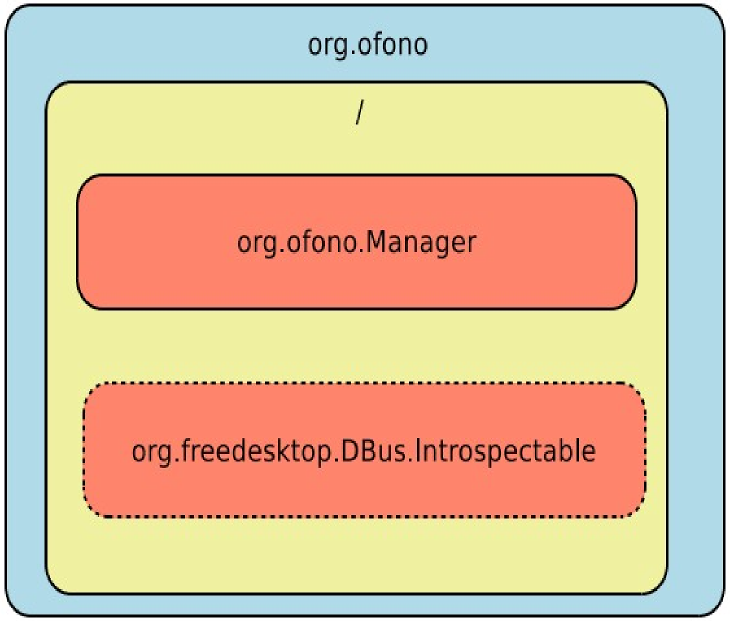
D-Bus Objects

- Are attached to one service
- Can be dynamically created or removed
- Are uniquely identified by an **object path** (ie / or `/net/connman/technology/cellular`)
- Implement one or several interfaces

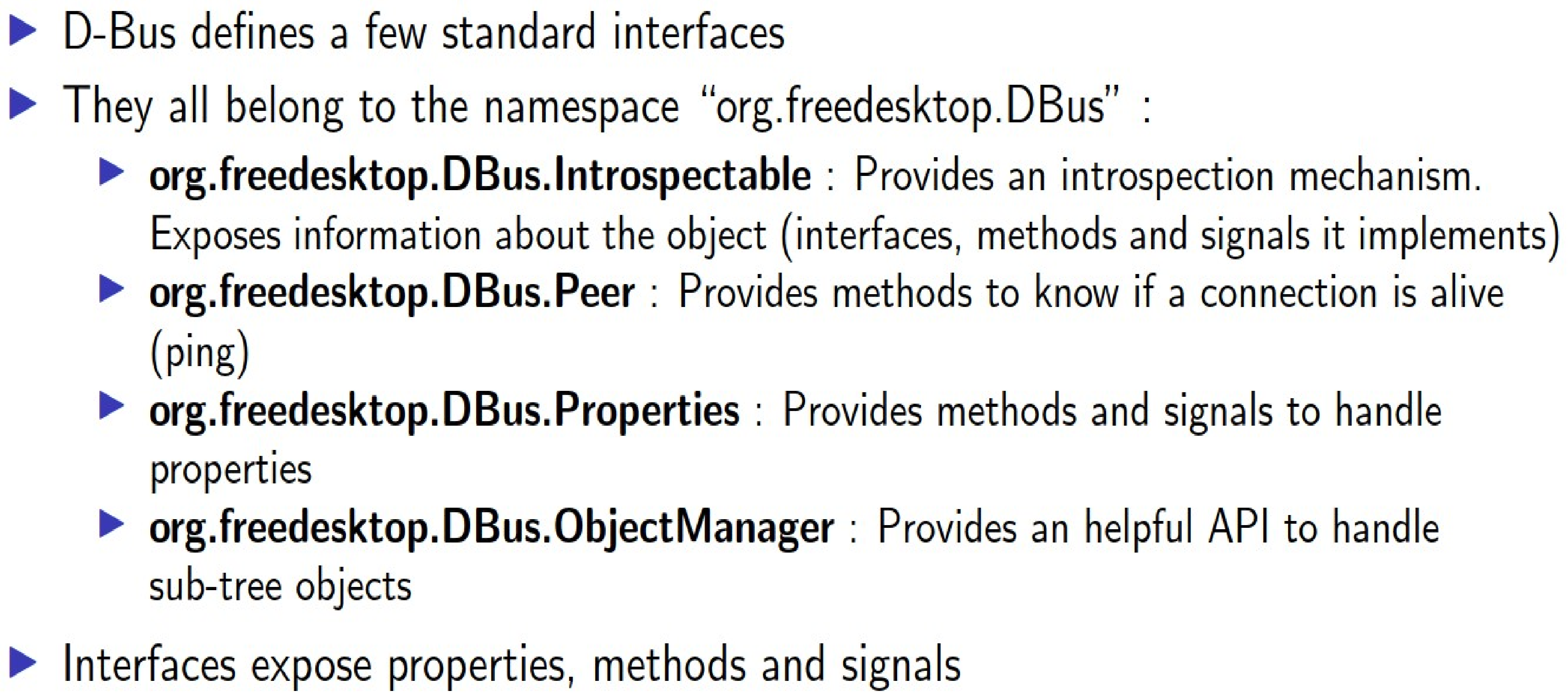


D-Bus Interfaces

- Can be compared to a “namespace” in Java
- Has a unique name ressembling Java interface names, using dots (ie `org.ofono.Manager`)
- Contains *members*: properties, methods and signals



D-Bus Interfaces



- D-Bus defines a few standard interfaces
- They all belong to the namespace "org.freedesktop.DBus" :
  - **org.freedesktop.DBus.Introspectable** : Provides an introspection mechanism. Exposes information about the object (interfaces, methods and signals it implements)
  - **org.freedesktop.DBus.Peer** : Provides methods to know if a connection is alive (ping)
  - **org.freedesktop.DBus.Properties** : Provides methods and signals to handle properties
  - **org.freedesktop.DBus.ObjectManager** : Provides an helpful API to handle sub-tree objects
- Interfaces expose properties, methods and signals

D-Bus Properties

- Directly accessible fields
- Can be read / written
- Can be of different types defined by the D-Bus specification :
  - basic types: bytes, boolean, integer, double, ...
  - string-like types : string, object path (must be valid) and signature
  - container-types: structure, array, variant (complex types) and dictionnary entry (hash)
- Very convenient standard interface : `org.freedesktop.DBus.Properties`
- Types are represented by characters

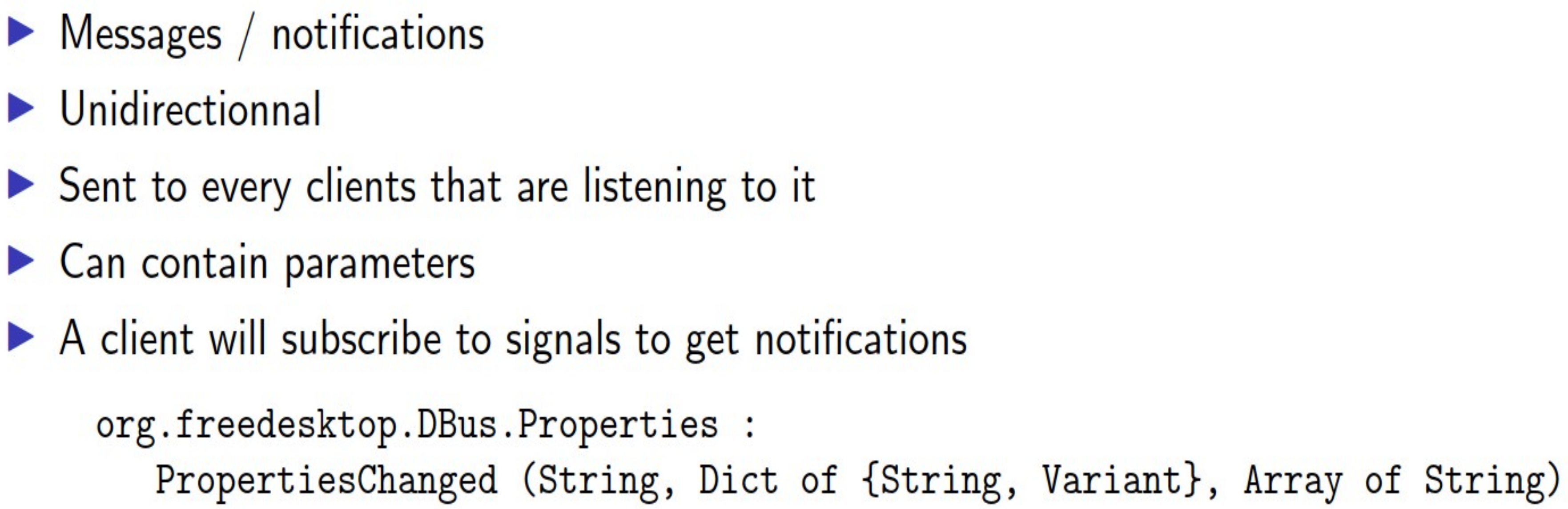
byte	y	string	s	variant	v
boolean	b	object-path	o	array of int32	ai
int32	i	array	a	array of an array of int32	aai
uint32	u	struct	()	array of a struct with 2 int32 fields	a(ii)
double	d	dict	{}	dict of string and int32	{si}

D-Bus Methods

- allow remote procedure calls from one process to another
- Can be passed one or several parameters
- Can return values/objects
- Look like any method you could know from other languages

```
org.freedesktop.DBus.Properties :
  Get (String interface_name, String property_name) => Variant value
  GetAll (String interface_name) => Dict of {String, Variant} props
  Set (String interface_name, String property_name, Variant value)
```

D-Bus Signals



- Messages / notifications
- Unidirectionnal
- Sent to every clients that are listening to it
- Can contain parameters
- A client will subscribe to signals to get notifications

```
org.freedesktop.DBus.Properties :
   PropertiesChanged (String, Dict of {String, Variant}, Array of String)
```

D-Bus Policy

- Adds a security mechanism
- Represented by XML files
- Handled by each `dbus-daemon` (under `/etc/dbus-1/session.d` and `/etc/dbus-1/system.d`)
- Allows the administrator to control which user can talk to which interface, which user can send message to which interface, and so on
- If you are not able to talk with a D-Bus service or get an `org.freedesktop.DBus.Error.AccessDenied` error, check this file!
- `org.freedesktop.PolicyKit1` has been created to handle all security accesses

D-Bus

- In this example, "toto" can :
  - own the interface `org.ofono`
  - send messages to the owner of the given service
  - call `GetContexts` from interface `org.ofono.ConnectionManager`

```
<!DOCTYPE busconfig PUBLIC
 ``-//freedesktop//DTD D-BUS Bus Configuration 1.0//EN''
 ``http://www.freedesktop.org/standards/dbus/1.0/busconfig.dtd''>
 <busconfig>
    <policy user="toto">
        <allow own="org.ofono"/>
        <allow send_destination="org.ofono"/>
        <allow send_interface="org.ofono.ConnectionManager" send_member="GetContexts"/>
    </policy>
</busconfig>
```

- Can `allow` or `deny`

D-Bus

- Libdbus
  - This is the low-level library used by the dbus-daemon.
  - As the homepage of the project says: *"If you use this low-level API directly, you're signing up for some pain"*.
  - Recommended to use it only for small programs and you do not want to add many dependencies
- GDbus
  - Is part of GLib (GIO)
  - Provides a very comfortable API
- QtDbus
  - Is a Qt module
  - Is useful if you already have Qt on your system
  - Contains many classes to handle/interact such as `QDBusInterface`

D-Bus

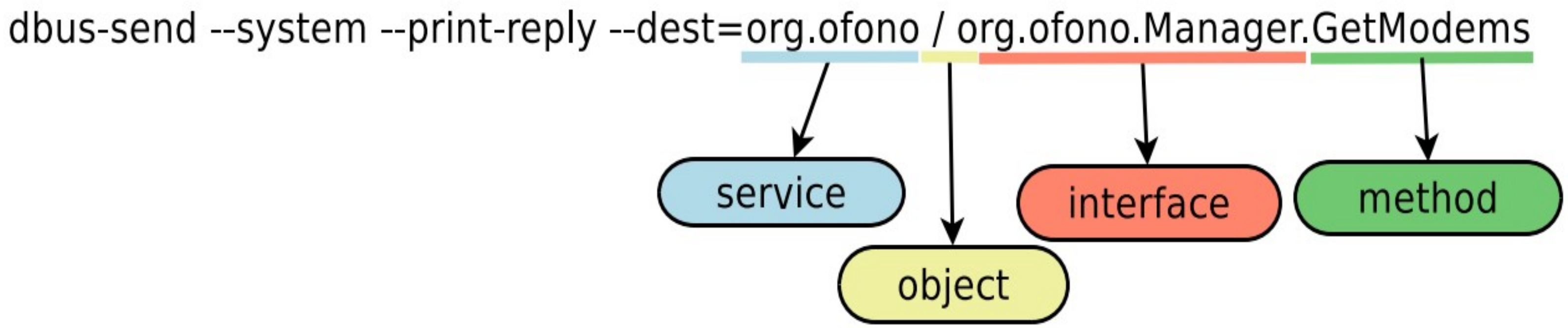
- Bindings exist for other languages: dbus-python, dbus-java, ...
- All the bindings allow to:
  - Interact with existing D-Bus services
  - Create your own D-Bus services, objects, interfaces, and so on!
  - but... D-Bus is not a high performance IPC
  - Should be used only for **control** and not data
  - For example, you can use it to activate an audio pipeline but not to send the audio stream

D-Bus

- Will present every tool with a demo
- dbus-send: Command-line interface (cli) to call method of interfaces (and get/set properties)
- dbus-monitor: Cli to subscribe and monitor signals
- gdbus: A GLib implementation of a more complete tool than dbus-send/monitor
- d-feet: A GUI application to handle all D-Bus services
- and others...

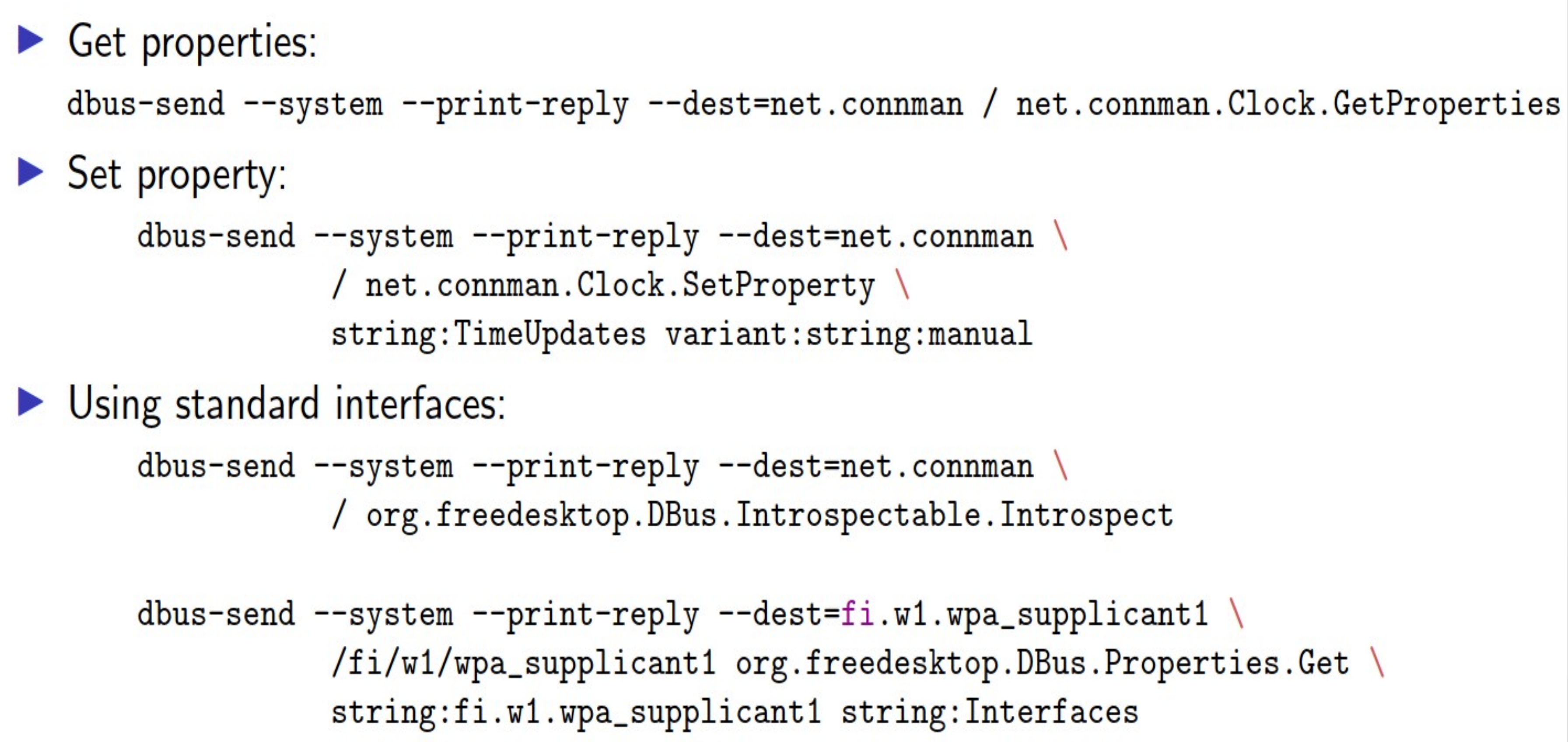
D-Bus

- Can chose the session or system bus (`--session` or `--system`)
- Here is an example:



```
dbus-send --system --print-reply --dest=org.ofono / org.ofono.Manager.GetModems
```

D-Bus



- Get properties:

```
dbus-send --system --print-reply --dest=net.connman / net.connman.Clock.GetProperties
```

- Set property:

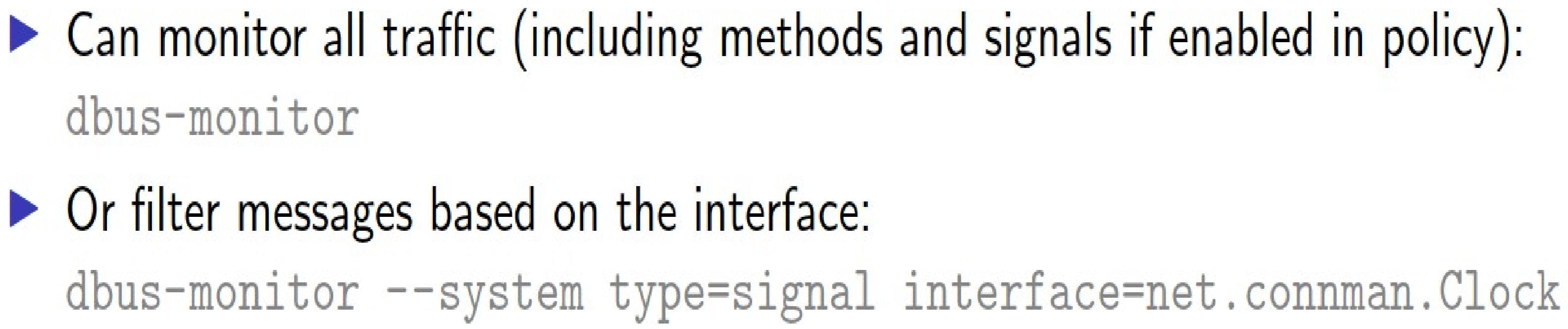
```
    dbus-send --system --print-reply --dest=net.connman \
              / net.connman.Clock.SetProperty \
              string:TimeUpdates variant:string:manual
```

- Using standard interfaces:

```
    dbus-send --system --print-reply --dest=net.connman \
              / org.freedesktop.DBus.Introspectable.Introspect

    dbus-send --system --print-reply --dest=fi.w1.wpa_supplicant1 \
              /fi/w1/wpa_supplicant1 org.freedesktop.DBus.Properties.Get \
              string:fi.w1.wpa_supplicant1 string:Interfaces
```

D-Bus



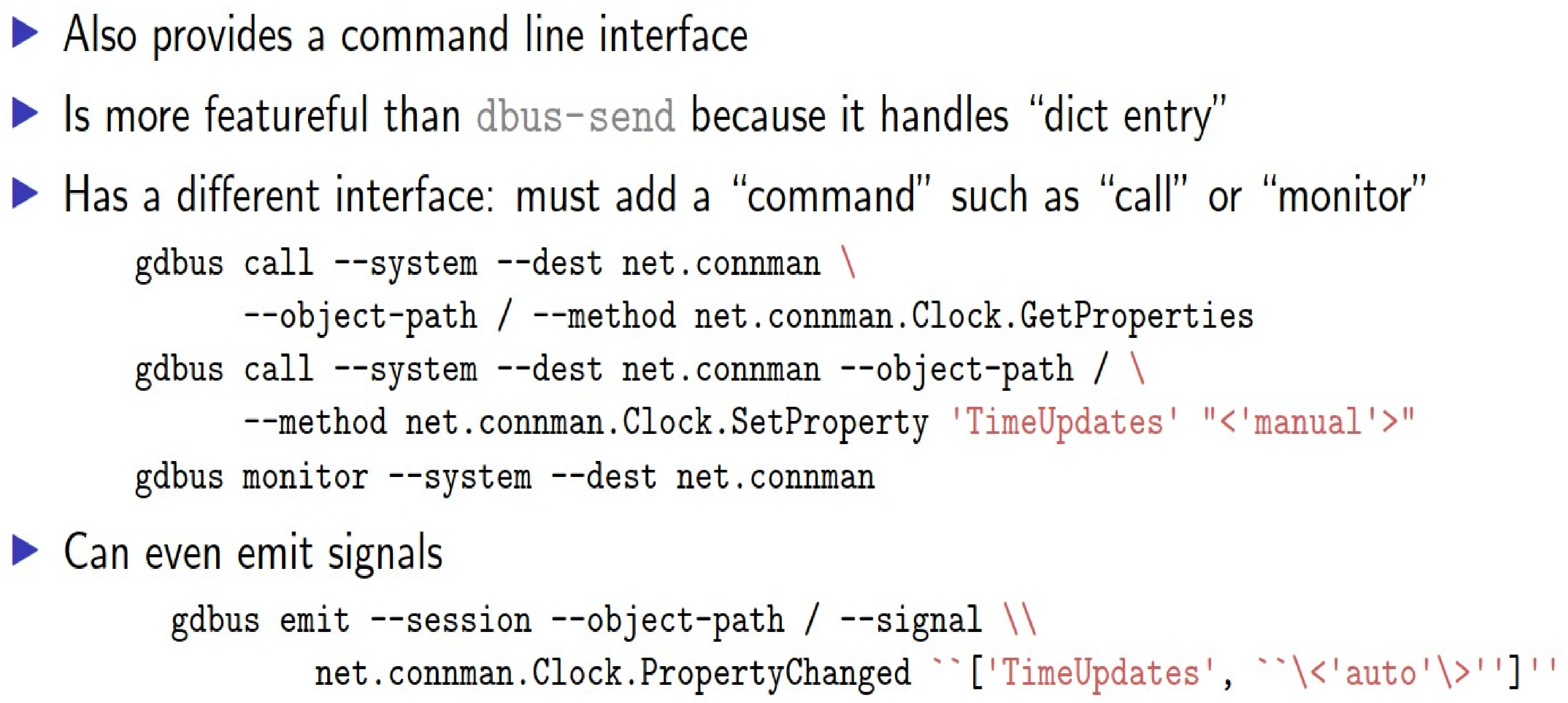
- Can monitor all traffic (including methods and signals if enabled in policy):

```
dbus-monitor
```

- Or filter messages based on the interface:

```
dbus-monitor --system type=signal interface=net.connman.Clock
```

D-Bus



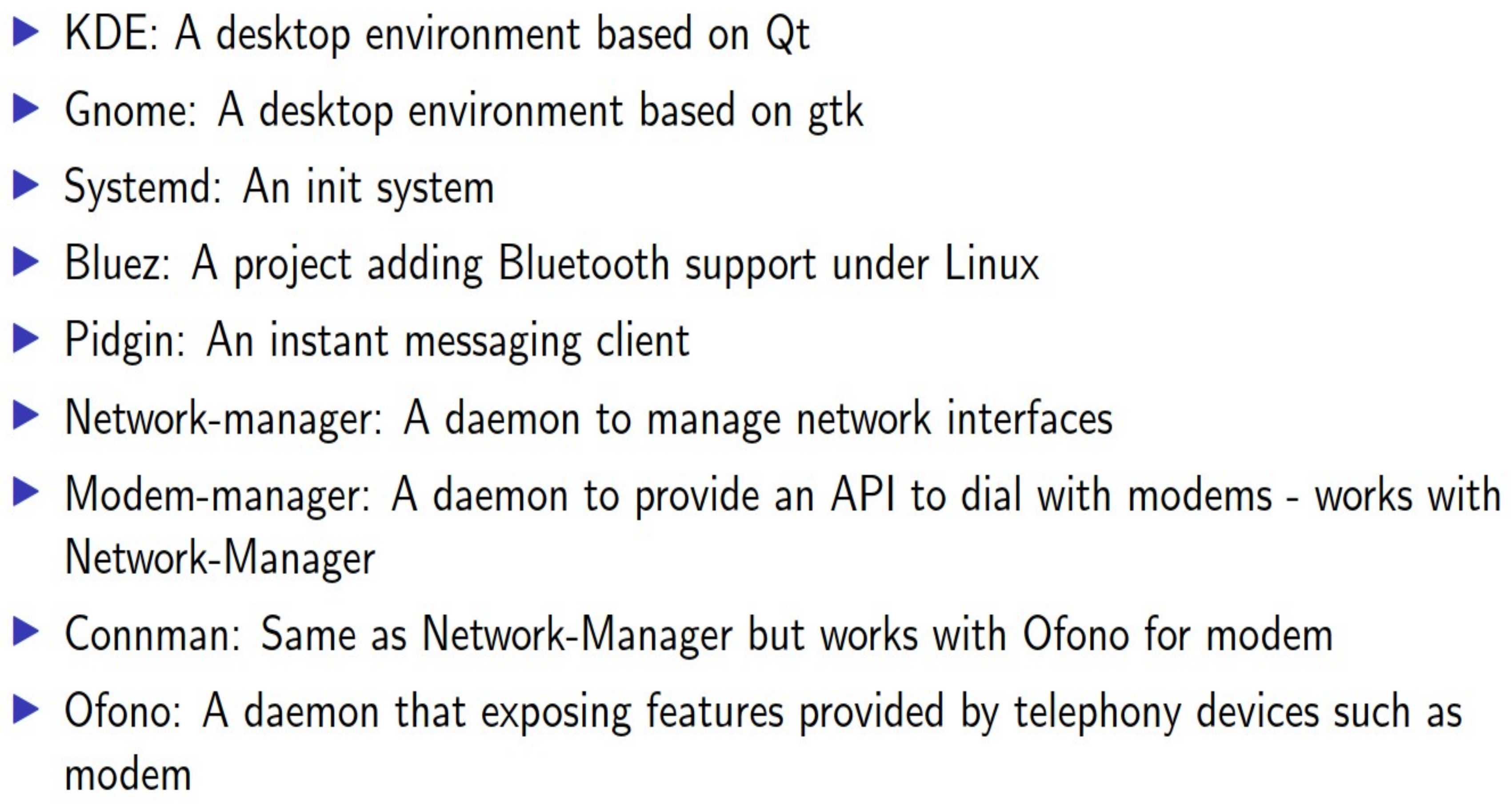
- Also provides a command line interface
- Is more featureful than `dbus-send` because it handles “dict entry”
- Has a different interface: must add a “command” such as “call” or “monitor”

```
gdbus call --system --dest net.connman \
      --object-path / --method net.connman.Clock.GetProperties
gdbus call --system --dest net.connman --object-path / \
      --method net.connman.Clock.SetProperty 'TimeUpdates' "<'manual'>"
gdbus monitor --system --dest net.connman
```

- Can even emit signals

```
gdbus emit --session --object-path / --signal \\
       net.connman.Clock.PropertyChanged ``['TimeUpdates', ``\<'auto'\>'']''
```

D-Bus



- KDE: A desktop environment based on Qt
- Gnome: A desktop environment based on gtk
- Systemd: An init system
- Bluez: A project adding Bluetooth support under Linux
- Pidgin: An instant messaging client
- Network-manager: A daemon to manage network interfaces
- Modem-manager: A daemon to provide an API to dial with modems - works with Network-Manager
- Connman: Same as Network-Manager but works with Ofono for modem
- Ofono: A daemon that exposing features provided by telephony devices such as modem