

KBUS

A simple messaging system

Tony Ibbs

`tibs@tonyibbs.co.uk`

`http://code.google.com/p/kbus/`

KBUS

- A Linux kernel module
- File based:
 - `/dev/kbus0`, etc.
 - `open`, `close`, `read`, `write`, `ioctl`
- Use it:
 - directly
 - via C library
 - via Python API
- Tested using the Python API

With thanks to



Two parts

- Simple use of KBUS
- Why KBUS

Simple use: Senders and Listeners

Terminal I: Rosencrantz

```
Python 2.6.4 (r264:75706, Dec  7 2009, 18:45:15)  
[GCC 4.4.1] on linux2  
Type "help", "copyright", "credits" or "license" for  
more information.  
>>> from kbus import Ksock
```

Terminal I: Rosencrantz

```
Python 2.6.4 (r264:75706, Dec  7 2009, 18:45:15)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for
more information.
>>> from kbus import Ksock
>>> rosenkrantz = Ksock(0)
>>> print rosenkrantz
Ksock device 0, id 113, mode read/write
```

Terminal I: Rosencrantz

```
Python 2.6.4 (r264:75706, Dec  7 2009, 18:45:15)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for
more information.
>>> from kbus import Ksock
>>> rosenkrantz = Ksock(0)
>>> print rosenkrantz
Ksock device 0, id 113, mode read/write
```


Terminal I: Rosencrantz

```
>>> from kbus import Message
>>> ahem = Message('$.Actor.Speak', 'Ahem')
>>> rosenkrantz.send_msg(ahem)
MessageId(0, 337)
```

Terminal I: Rosencrantz

```
>>> from kbus import Message
>>> ahem = Message('$.Actor.Speak', 'Ahem')
>>> rosenkrantz.send_msg(ahem)
MessageId(0, 337)
```

Terminal I: Rosencrantz

```
>>> from kbus import Message
>>> ahem = Message('$.Actor.Speak', 'Ahem')
>>> rosenkrantz.send_msg(ahem)
MessageId(0, 337)
```

Terminal 2: Audience

```
Python 2.6.4 (r264:75706, Dec  7 2009, 18:45:15)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for
more information.
>>> from kbus import *
>>> audience = Ksock(0)
>>> audience.bind('$Actor.Speak')
```

Terminal 2: Audience

```
Python 2.6.4 (r264:75706, Dec  7 2009, 18:45:15)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for
more information.
>>> from kbus import *
>>> audience = Ksock(0)
>>> audience.bind('$.Actor.Speak')
```

Terminal 2: Audience

```
Python 2.6.4 (r264:75706, Dec  7 2009, 18:45:15)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for
more information.
>>> from kbus import *
>>> audience = Ksock(0)
>>> audience.bind('$.Actor.Speak')
```

Terminal I: Rosencrantz

```
>>> rosenkrantz.send_msg(ahem)  
MessageId(0, 338)
```

Terminal 1: Rosencrantz

```
>>> rosenkrantz.send_msg(ahem)  
MessageId(0, 338)
```

Terminal 2: Audience

```
>>> audience.read_next_msg()  
Message('$.Actor.Speak', data='Ahem', from_=113L,  
id=MessageId(0, 338))
```


Terminal 1: Rosencrantz

```
>>> rosenkrantz.send_msg(ahem)  
MessageId(0, 338)
```

Terminal 2: Audience

```
>>> audience.read_next_msg()  
Message('$.Actor.Speak', data='Ahem', from_=113L,  
id=MessageId(0, 338))
```

Terminal 1: Rosencrantz

```
>>> rosenkrantz.send_msg(ahem)  
MessageId(0, 338)
```

Terminal 2: Audience

```
>>> audience.read_next_msg()  
Message('$.Actor.Speak', data='Ahem', from_=113L,  
id=MessageId(0, 338))
```

Terminal 2: Audience

```
>>> print _  
<Announcement '$.Actor.Speak', id=[0:338], from=113,  
data='Ahem'>
```

Terminal 2: Audience

```
>>> print _  
<Announcement '$.Actor.Speak', id=[0:338], from=113,  
data='Ahem'>
```

Terminal 1: Rosencrantz

```
>>> rosenkrantz.ksock_id()  
113L
```

Terminal 2: Audience

```
>>> print audience.read_next_msg()  
None
```

Terminal 2: Audience

```
>>> import select
>>> while 1:
...     (r,w,x) = select.select([audience], [], [])
...     # At this point, r should contain audience
...     print audience.read_next_msg()
... 
```

Terminal I: Rosencrantz

```
>>> rosenkrantz.send_msg(Message('$Actor.Speak',  
... 'Hello there'))  
MessageId(0, 339)  
>>> rosenkrantz.send_msg(Message('$Actor.Speak',  
... 'Can you hear me?'))  
MessageId(0, 340)
```

Terminal 1: Rosencrantz

```
>>> rosenkrantz.send_msg(Message('$$.Actor.Speak',  
... 'Hello there'))  
MessageId(0, 339)  
>>> rosenkrantz.send_msg(Message('$$.Actor.Speak',  
... 'Can you hear me?'))  
MessageId(0, 340)
```

Terminal 2: Audience

```
<Announcement '$$.Actor.Speak', id=[0:339], from=113,  
data='Hello there'>  
<Announcement '$$.Actor.Speak', id=[0:340], from=113,  
data='Can you hear me?'>
```


Terminal 3: Guildenstern

```
Python 2.6.4 (r264:75706, Dec  7 2009, 18:45:15)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for
more information.
>>> from kbus import *
>>> guildenstern = Ksock(0)
>>> print guildenstern
Ksock device 0, id 115, mode read/write
```

Terminal 3: Guildenstern

```
>>> guildenstern.bind('$.Actor.*')
```

Terminal 3: Guildenstern

```
>>> guildenstern.bind('$.Actor.*')
```

Terminal 3: Guildenstern

```
>>> guildenstern.bind('$$.Actor.*')
```

Terminal 2: Audience

```
<CTRL-C>
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 2, in <module>
```

```
KeyboardInterrupt
```

```
>>> audience.bind('$$.Actor.*')
```

```
>>> while 1:
```

```
...     print audience.wait_for_msg()
```

```
...
```

Terminal 3: Guildenstern

```
>>> guildenstern.bind('$$.Actor.*')
```

Terminal 2: Audience

```
<CTRL-C>
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 2, in <module>
```

```
KeyboardInterrupt
```

```
>>> audience.bind('$$.Actor.*')
```

```
>>> while 1:
```

```
...     print audience.wait_for_msg()
```

```
...
```

Terminal I: Rosencrantz

```
>>> rosenkrantz.bind('$.Actor.*')
```

Terminal 3: Guildenstern

```
>>> guildenstern.send_msg(Message('$ .Actor.Speak',  
'Pssst!'))  
MessageId(0, 341)
```

Terminal 3: Guildenstern

```
>>> guildenstern.send_msg(Message('$.Actor.Speak',  
'Pssst!'))  
MessageId(0, 341)  
>>> print guildenstern.read_next_msg()  
<Announcement '$.Actor.Speak', id=[0:341], from=115,  
data='Pssst! '>
```


Terminal I: Rosencrantz

```
>>> msg = rosenkrantz.read_next_msg()
>>> print msg
<Announcement '$.Actor.Speak', id=[0:341], from=115,
data='Pssst! '>
```

Terminal 1: Rosencrantz

```
>>> msg = rosenkrantz.read_next_msg()
>>> print msg
<Announcement '$.Actor.Speak', id=[0:341], from=115,
data='Pssst! '>
```

Terminal 2: Audience

```
<Announcement '$.Actor.Speak', id=[0:341], from=115,
data='Pssst! '>
<Announcement '$.Actor.Speak', id=[0:341], from=115,
data='Pssst! '>
```

Terminal 2: Audience

<CTRL-C>

Traceback (most recent call last):

File "<stdin>", line 2, in <module>

File ".../ksock.py", line 492, in wait_for_msg

(r, w, x) = select.select([self], [], [], timeout)

KeyboardInterrupt

Terminal 2: Audience

<CTRL-C>

Traceback (most recent call last):

File "<stdin>", line 2, in <module>

File ".../ksock.py", line 492, in wait_for_msg

(r, w, x) = select.select([self], [], [], timeout)

KeyboardInterrupt

>>> audience.unbind('\$Actor.Speak')

Terminal 2: Audience

<CTRL-C>

Traceback (most recent call last):

File "<stdin>", line 2, in <module>

File ".../ksock.py", line 492, in wait_for_msg

(r, w, x) = select.select([self], [], [], timeout)

KeyboardInterrupt

>>> audience.unbind('\$Actor.Speak')

Terminal 2: Audience

<CTRL-C>

Traceback (most recent call last):

File "<stdin>", line 2, in <module>

File ".../ksock.py", line 492, in wait_for_msg

(r, w, x) = select.select([self], [], [], timeout)

KeyboardInterrupt

>>> audience.unbind('\$Actor.Speak')

Terminal 2: Audience

<CTRL-C>

Traceback (most recent call last):

File "<stdin>", line 2, in <module>

File ".../ksock.py", line 492, in wait_for_msg

(r, w, x) = select.select([self], [], [], timeout)

KeyboardInterrupt

```
>>> audience.unbind('$Actor.Speak')
```

```
>>> while 1:
```

```
...     msg = audience.wait_for_msg()
```

```
...     print msg
```

```
...
```

Simple use: Requests and Repliers

Terminal 3: Guildenstern

```
>>> guildenstern.bind('$.Actor.Ask.Guildenstern', True)
```

Terminal I: Rosencrantz

```
>>> from kbus import Request
>>> req = Request('$.Actor.Ask.Guildenstern',
... 'Were you speaking to me?')
>>> rosenkrantz.send_msg(req)
MessageId(0, 342)
```

Terminal I: Rosencrantz

```
>>> from kbus import Request
>>> req = Request('$.Actor.Ask.Guildenstern',
... 'Were you speaking to me?')
>>> rosenkrantz.send_msg(req)
MessageId(0, 342)
```

Terminal I: Rosencrantz

```
>>> print rosenkrantz.read_next_msg()  
<Request '$.Actor.Ask.Guildenstern', id=[0:342],  
from=113, flags=0x1 (REQ), data='Were you speaking to  
me? '>
```

Terminal I: Rosencrantz

```
>>> print rosenkrantz.read_next_msg()
<Request '$.Actor.Ask.Guildenstern', id=[0:342],
from=113, flags=0x1 (REQ), data='Were you speaking to
me? '>
>>> rosenkrantz.unbind('$.Actor.*')
```

Terminal 3: Guildenstern

```
>>> req = guildenstern.read_next_msg()
>>> print req
<Request '$.Actor.Ask.Guildenstern', id=[0:342],
from=113, flags=0x3 (REQ,YOU), data='Were you speaking
to me? '>
```

Terminal 3: Guildenstern

```
>>> req = guildenstern.read_next_msg()
>>> print req
<Request '$.Actor.Ask.Guildenstern', id=[0:342],
from=113, flags=0x3 (REQ,YOU), data='Were you speaking
to me? '>
```

Terminal 3: Guildenstern

```
>>> req = guildenstern.read_next_msg()
>>> print req
<Request '$.Actor.Ask.Guildenstern', id=[0:342],
from=113, flags=0x3 (REQ, YOU), data='Were you speaking
to me? '>
```


Terminal 3: Guildenstern

```
>>> req = guildenstern.read_next_msg()
>>> print req
<Request '$.Actor.Ask.Guildenstern', id=[0:342],
from=113, flags=0x3 (REQ, YOU), data='Were you speaking
to me? '>
>>> print req.wants_us_to_reply()
True
```

Terminal 3: Guildenstern

```
>>> req = guildenstern.read_next_msg()
>>> print req
<Request '$.Actor.Ask.Guildenstern', id=[0:342],
from=113, flags=0x3 (REQ, YOU), data='Were you speaking
to me? '>
>>> print req.wants_us_to_reply()
True
```

Terminal 3: Guildenstern

```
>>> msg = guildenstern.read_next_msg()
>>> print msg
<Request '$.Actor.Ask.Guildenstern', id=[0:342],
from=113, flags=0x1 (REQ), data='Were you speaking to
me? '>
```

Terminal 3:Guildenstern

```
>>> msg = guildenstern.read_next_msg()
>>> print msg
<Request '$.Actor.Ask.Guildenstern', id=[0:342],
from=113, flags=0x1 (REQ), data='Were you speaking to
me? '>
```

Terminal 3:Guildenstern

```
>>> msg = guildenstern.read_next_msg()
>>> print msg
<Request '$.Actor.Ask.Guildenstern', id=[0:342],
from=113, flags=0x1 (REQ), data='Were you speaking to
me? '>
>>> guildenstern.unbind('$.Actor.*')
```

Terminal 3: Guildenstern

```
>>> rep = reply_to(req, 'Yes, yes I was')
>>> print rep
<Reply '$.Actor.Ask.Guildenstern', to=113, in_reply_to=
[0:342], data='Yes, yes I was'>
```

Terminal 3: Guildenstern

```
>>> rep = reply_to(req, 'Yes, yes I was')  
>>> print rep  
<Reply '$.Actor.Ask.Guildenstern', to=113, in_reply_to=  
[0:342], data='Yes, yes I was'>
```

Terminal 3: Guildenstern

```
>>> rep = reply_to(req, 'Yes, yes I was')
>>> print rep
<Reply '$.Actor.Ask.Guildenstern', to=113, in_reply_to=
[0:342], data='Yes, yes I was'>
>>> guildenstern.send_msg(rep)
MessageId(0, 343)
```


Terminal 3: Guildenstern

```
>>> rep = reply_to(req, 'Yes, yes I was')
>>> print rep
<Reply '$.Actor.Ask.Guildenstern', to=113, in_reply_to=
[0:342], data='Yes, yes I was'>
>>> guildenstern.send_msg(rep)
MessageId(0, 343)
>>> guildenstern.read_next_msg()
```

Terminal I: Rosencrantz

```
>>> rep = rosenkrantz.read_next_msg()
>>> print rep
<Reply '$.Actor.Ask.Guildenstern', id=[0:343], to=113,
from=115, in_reply_to=[0:342], data='Yes, yes I was'>
```

Terminal 2: Audience

```
<Request '$.Actor.Ask.Guildenstern', id=[0:342],  
from=113, flags=0x1 (REQ), data='Were you speaking to  
me? '>
```

```
<Reply '$.Actor.Ask.Guildenstern', id=[0:343], to=113,  
from=115, in_reply_to=[0:342], data='Yes, yes I was'>
```

Simple use: Stateful Requests

Terminal I: Rosencrantz

```
>>> req = Request('$.Actor.Ask.Guildenstern',  
>>> 'Will you count heads for me?')  
>>> rosenkrantz.send_msg(req)  
MessageId(0, 343)
```

Terminal 3: Guildenstern

```
>>> req = guildenstern.read_next_msg()  
>>> guildenstern.send_msg(reply_to(req, 'I shall'))  
MessageId(0, 345)
```

Terminal 3: Guildenstern

```
>>> req = guildenstern.read_next_msg()
>>> guildenstern.send_msg(reply_to(req, 'I shall'))
MessageId(0, 345)
>>> guildenstern.bind('$.Actor.CoinToss', True)
>>> heads = 0
>>> while True:
...     toss = guildenstern.wait_for_msg()
...     print toss
...     if toss.data == 'Head':
...         print 'A head - amazing'
...         heads += 1
...     else:
...         print 'Bah, tails'
...     rep = reply_to(toss, 'Head count is %d'%heads)
...     guildenstern.send_msg(rep)
... 
```

Terminal I: Rosencrantz

```
>>> rep = rosenkrantz.read_next_msg()  
>>> print rep.from_  
115  
>>> # Throws a head
```


Terminal I: Rosencrantz

```
>>> rep = rosenkrantz.read_next_msg()
>>> print rep.from_
115
>>> # Throws a head
```

Terminal I: Rosencrantz

```
>>> rep = rosenkrantz.read_next_msg()
```

```
>>> print rep.from_
```

```
115
```

```
>>> # Throws a head
```

Terminal I: Rosencrantz

```
>>> rep = rosenkrantz.read_next_msg()
>>> print rep.from_
115
>>> # Throws a head
... from kbus import stateful_request
>>> sreq = stateful_request(rep, '$.Actor.CoinToss',
... 'Head')
>>> print sreq
<Request '$.Actor.CoinToss', to=115, flags=0x1 (REQ),
data='Head'>
```

Terminal I: Rosencrantz

```
>>> rep = rosenkrantz.read_next_msg()
>>> print rep.from_
115
>>> # Throws a head
... from kbus import stateful_request
>>> sreq = stateful_request(rep, '$.Actor.CoinToss',
... 'Head')
>>> print sreq
<Request '$.Actor.CoinToss', to=115, flags=0x1 (REQ),
data='Head'>
```

Terminal I: Rosencrantz

```
>>> rep = rosenkrantz.read_next_msg()
>>> print rep.from_
115
>>> # Throws a head
... from kbus import stateful_request
>>> sreq = stateful_request(rep, '$.Actor.CoinToss',
... 'Head')
>>> print sreq
<Request '$.Actor.CoinToss', to=115, flags=0x1 (REQ),
data='Head'>
```

Terminal I: Rosencrantz

```
>>> rep = rosenkrantz.read_next_msg()
>>> print rep.from_
115
>>> # Throws a head
... from kbus import stateful_request
>>> sreq = stateful_request(rep, '$.Actor.CoinToss',
... 'Head')
>>> print sreq
<Request '$.Actor.CoinToss', to=115, flags=0x1 (REQ),
data='Head'>
```

Terminal I: Rosencrantz

```
>>> rep = rosenkrantz.read_next_msg()
>>> print rep.from_
115
>>> # Throws a head
... from kbus import stateful_request
>>> sreq = stateful_request(rep, '$.Actor.CoinToss',
... 'Head')
>>> print sreq
<Request '$.Actor.CoinToss', to=115, flags=0x1 (REQ),
data='Head'>
>>> rosenkrantz.send_msg(sreq)
MessageId(0, 346)
```

Terminal 3: Guildenstern

```
<Request '$.Actor.CoinToss', id=[0:346], to=115,  
from=113, flags=0x3 (REQ,YOU), data='Head'>
```

```
A head - amazing
```

```
MessageId(0, 347)
```


Terminal 3: Guildenstern

```
<Request '$.Actor.CoinToss', id=[0:346], to=115,  
from=113, flags=0x3 (REQ,YOU), data='Head'>  
A head - amazing  
MessageId(0, 347)
```

Terminal 1: Rosencrantz

```
>>> count = rosenkrantz.read_next_msg()  
>>> print 'So,',count.data  
So, Head count is 1  
>>> # Throws a head  
... sreq = stateful_request(rep, '$.Actor.CoinToss',  
... 'Head')  
>>> rosenkrantz.send_msg(sreq)  
MessageId(0, 348)
```

Terminal 3: Guildenstern

```
<Request '$.Actor.CoinToss', id=[0:346], to=115,  
from=113, flags=0x3 (REQ,YOU), data='Head'>  
A head - amazing  
MessageId(0, 347)
```

Terminal 1: Rosencrantz

```
>>> count = rosenrantz.read_next_msg()  
>>> print 'So,',count.data  
So, Head count is 1  
>>> # Throws a head  
... sreq = stateful_request(rep, '$.Actor.CoinToss',  
... 'Head')  
>>> rosenrantz.send_msg(sreq)  
MessageId(0, 348)
```

Terminal 3: Guildenstern

```
<Request '$.Actor.CoinToss', id=[0:348], to=115,  
from=113, flags=0x3 (REQ,YOU), data='Head'>
```

```
A head - amazing
```

```
MessageId(0, 349)
```

Terminal I: Rosencrantz

```
>>> count = rosenkrantz.read_next_msg()
>>> print 'So,',count.data
So, Head count is 2
>>> # Throws a head
```

Terminal 3: Guildenstern

<CTRL-C>

Traceback (most recent call last):

File "<stdin>", line 2, in <module>

File ".../ksock.py", line 492, in wait_for_msg

(r, w, x) = select.select([self], [], [], timeout)

KeyboardInterrupt

Terminal 3: Guildenstern

<CTRL-C>

Traceback (most recent call last):

File "<stdin>", line 2, in <module>

File ".../ksock.py", line 492, in wait_for_msg

(r, w, x) = select.select([self], [], [], timeout)

KeyboardInterrupt

```
>>> print 'Falstaff! No! Ouch!'
```

```
Falstaff! No! Ouch!
```

```
>>> guildenstern.close()
```

Terminal 4: Falstaff

```
Python 2.6.4 (r264:75706, Dec  7 2009, 18:45:15)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for
more information.
>>> from kbus import *
>>> falstaff = Ksock(0)
>>> falstaff.bind('$.Actor.CoinToss', True)
```

Terminal I: Rosencrantz

```
... sreq = stateful_request(rep, '$.Actor.CoinToss',  
... 'Head')  
>>> rosenkrantz.send_msg(sreq)  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
  File ".../ksock.py", line 432, in send_msg  
    return self.send()  
  File ".../ksock.py", line 220, in send  
    fcntl.ioctl(self.fd, Ksock.IOC_SEND, arg);  
IOError: [Errno 32] Broken pipe
```


Terminal I: Rosencrantz

```
... sreq = stateful_request(rep, '$.Actor.CoinToss',
... 'Head')
>>> rosenkrantz.send_msg(sreq)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File ".../ksock.py", line 432, in send_msg
    return self.send()
  File ".../ksock.py", line 220, in send
    fcntl.ioctl(self.fd, Ksock.IOC_SEND, arg);
IOError: [Errno 32] Broken pipe
```

Terminal I: Rosencrantz

```
... sreq = stateful_request(rep, '$.Actor.CoinToss',  
... 'Head')
```

```
>>> rosenkrantz.send_msg(sreq)
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
File ".../ksock.py", line 432, in send_msg
```

```
    return self.send()
```

```
File ".../ksock.py", line 220, in send
```

```
    fcntl.ioctl(self.fd, Ksock.IOC_SEND, arg);
```

```
IOError: [Errno 32] Broken pipe
```

```
$ errno.py 32
```

```
Error 32 (0x20) is EPIPE: Broken pipe
```

KBUS:

On attempting to send 'to' a specific replier, the replier with that id is no longer bound to the given message's name.

Terminal 2: Audience

```
<Request '$.Actor.Ask.Guildenstern', id=[0:344],  
from=113, flags=0x1 (REQ), data='Will you count heads  
for me? '>  
<Reply '$.Actor.Ask.Guildenstern', id=[0:345], to=113,  
from=115, in_reply_to=[0:344], data='Yes, yes I shall'>  
<Request '$.Actor.CoinToss', id=[0:346], to=115,  
from=113, flags=0x1 (REQ), data='Head'>  
<Reply '$.Actor.CoinToss', id=[0:347], to=113,  
from=115, in_reply_to=[0:346], data='Head count is 1'>  
<Request '$.Actor.CoinToss', id=[0:348], to=115,  
from=113, flags=0x1 (REQ), data='Head'>  
<Reply '$.Actor.CoinToss', id=[0:349], to=113,  
from=115, in_reply_to=[0:348], data='Head count is 2'>
```

Why KBUS

Why

- We work in the embedded world
- We want a means of communication between software elements
- We've had experience of bad solutions

Bad things

- Race conditions when either end restarts
- Unreliability
- Poor documentation

Aims

- Simple models to “think with”
- Predictable delivery
- *Always* get a reply to a request
- Deterministic message order on a bus
- Small codebase, in C
- Easy to use from Python (well, I want that)
- Open source

Simple models: naming

- Ksock
- Sender, Listener, Replier
- Message, Announcement, Request, Reply
- “\$.message.name”

Simple models: APIs

- The “bare Unix” level
- The C library - hides the details
- The Python API - even better

Simple models: Data

KBUS does not say anything about the data being transferred

Predictable delivery

- It is acceptable for a Listener to miss messages
(although they should be able to avoid it)
- But it is not acceptable for a Replier to miss a Request
- And each Request *shall* produce a Reply

A “send” fails if:

- the sender of a Request has a full queue
(-ENOLCK)
- the receiver of a Request has a full queue
(-EBUSY)
- a message is marked “ALL or FAIL” and any of the listeners could not receive it
(-EBUSY)
- a message is marked “ALL or WAIT” and any of the listeners could not receive it
(-EAGAIN)

""KBUS guarantees that each Request will (eventually) be matched by a consequent Reply (or Status) message, and only one such.""

If the replier can't give a Reply, KBUS will generate one - for instance:

- “\$.KBUS.Replier.Unbound” or
- “\$.KBUS.Replier.GoneAway”

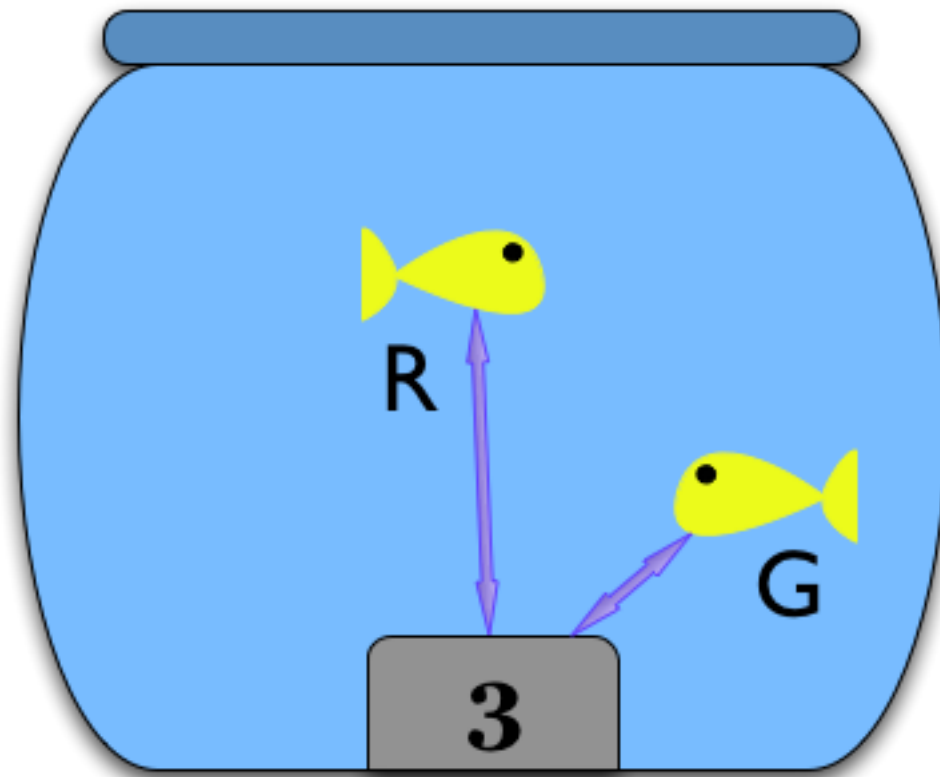
Kernel module

- we can have a reliable file interface
- but the kernel simplifies it for us
- guaranteed to know when a listener goes away (Ksock closes)
- realistic expectation of reliability
- kernel datastructures
- kernel memory handling

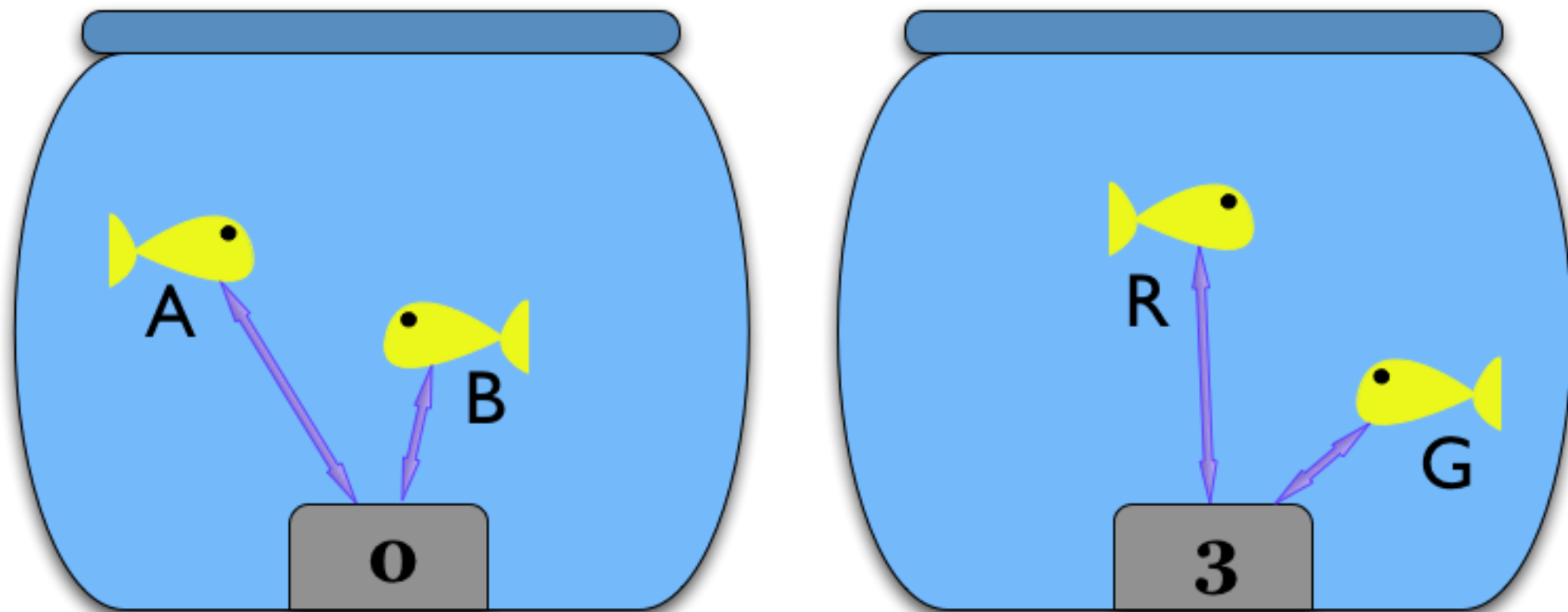
The kernel does stuff for us

- Requires predictable interfaces
- Enforces a coding style
- We can “ignore” threading, multiple CPUs, etc
- We should have less context switching
- We can try submitting it

Isolation



Each KBUS is isolated from the others, as if it were in a metaphorical goldfish bowl.



Two other fish, communicating via a different KBUS device, are in a different metaphorical bowl, and thus cannot communicate with R and G.

Example uses

- Set Top Box - Instructions from user interface to:
 - receiver (change channel, volume)
 - recorder (play, rewind, pause, record)
 - DRM interface
- Industrial control systems
- Remote control and telemetry

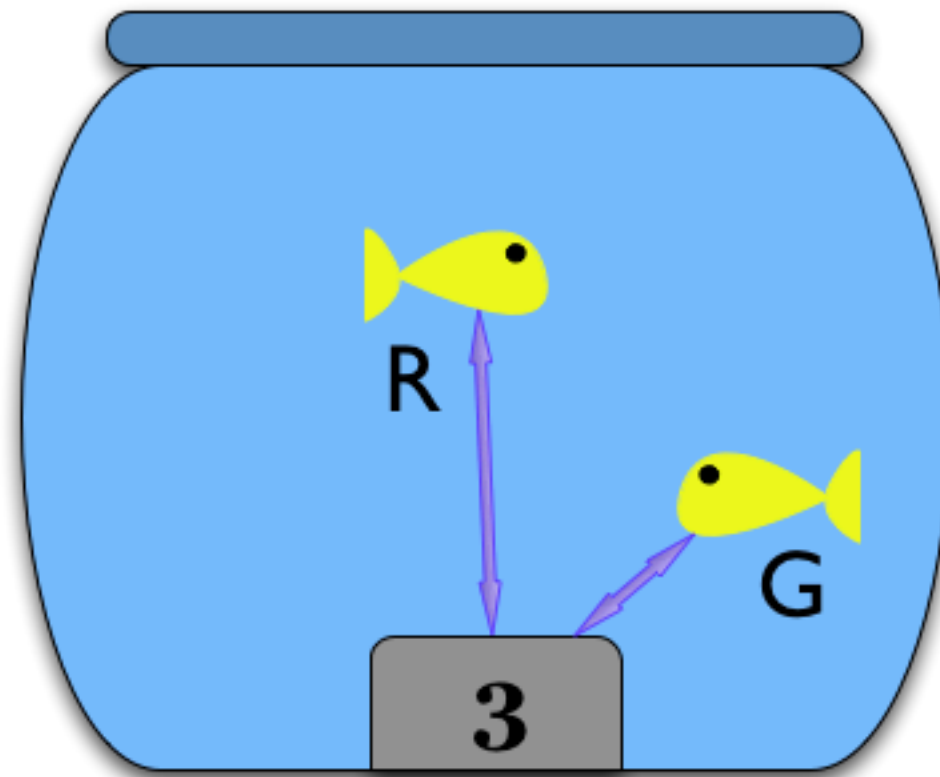
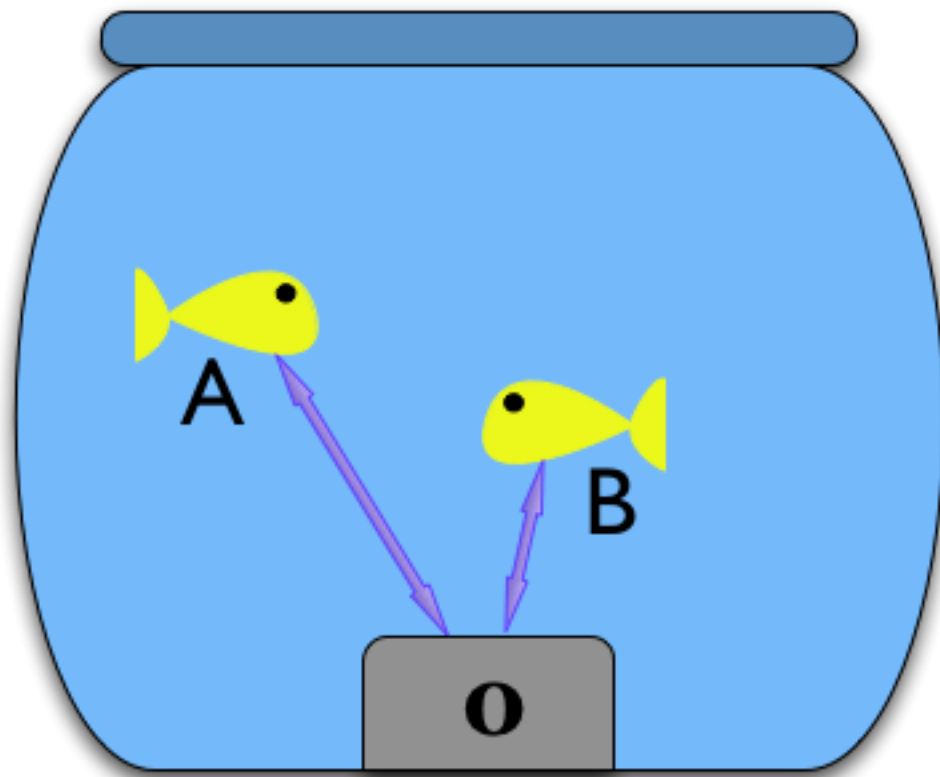
Any alternatives?

- POSIX message queues (mqqueue)
(new in 2.6.2, limited resource usage, too simple)
- DBUS
(complex, no message ordering, large)
- zeromq (0mq)
(pretty, pragmatic, C++, deliberately “simpler”)
- what else?

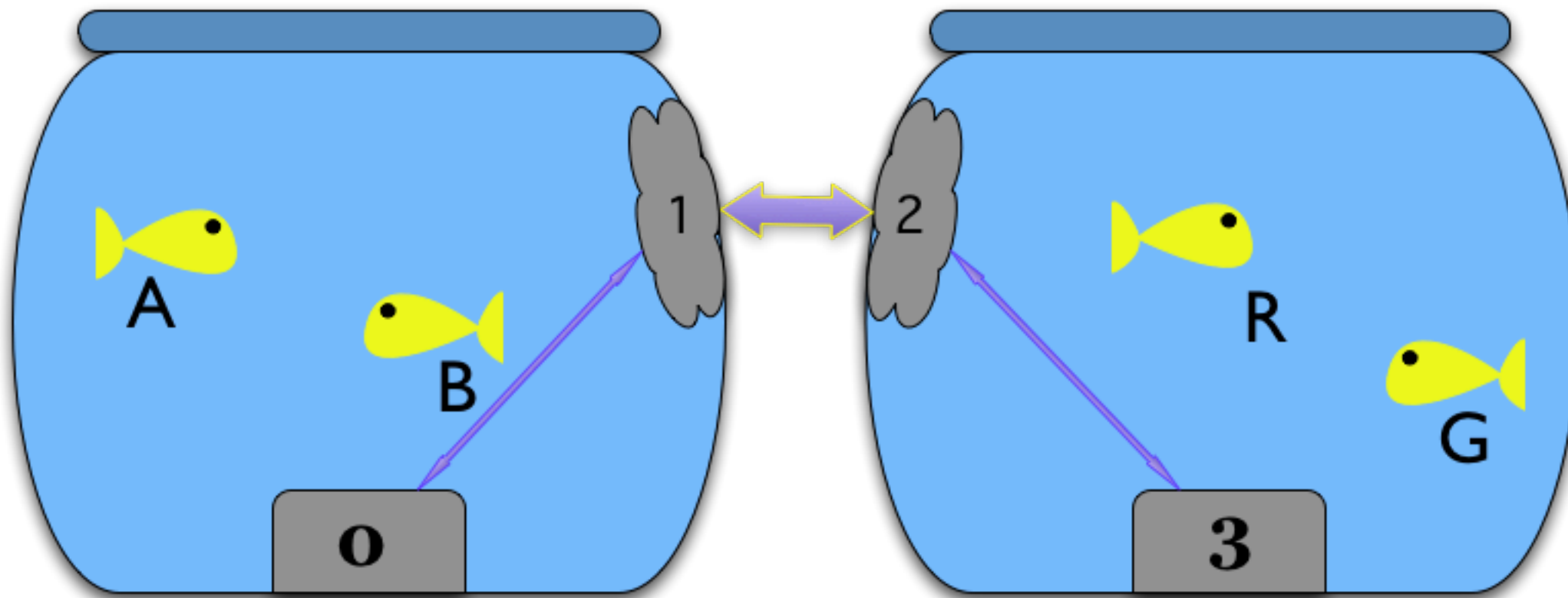
Fin

OK, Limpets...

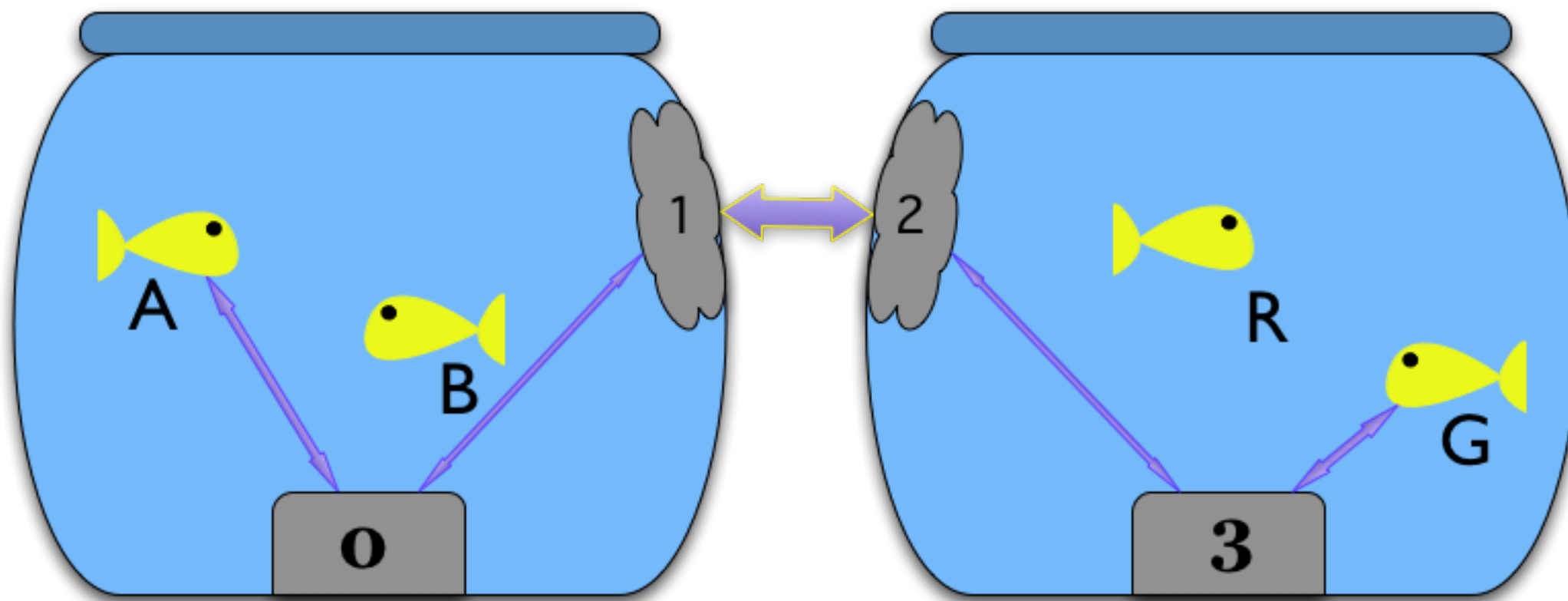
A Limpet proxies KBUS messages between a Ksock and another Limpet.



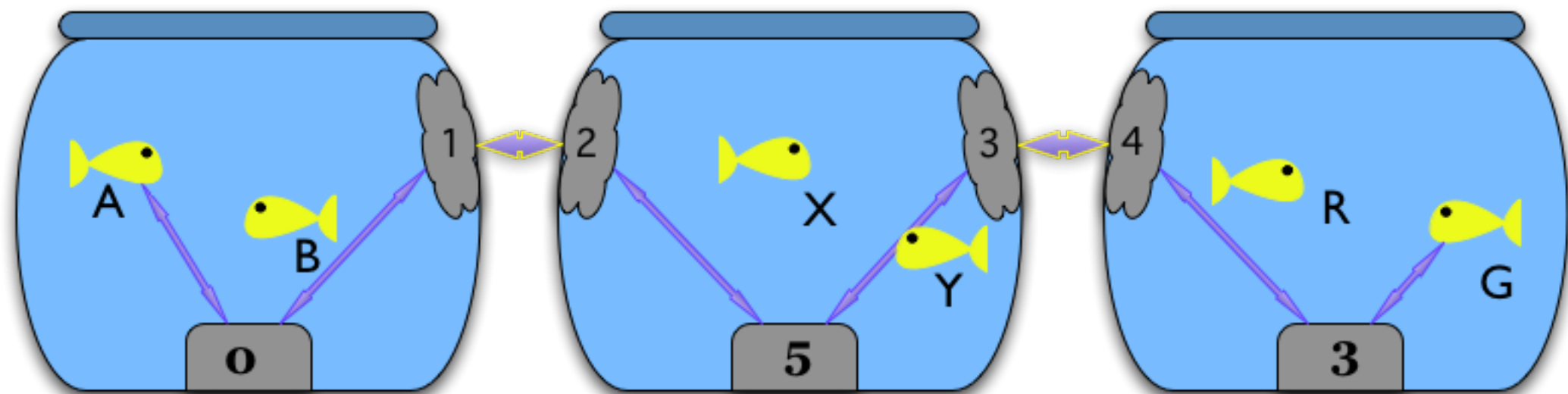
Our isolated bowls



A pair of Limpets



A talks to G



Even with intermediate
bowls

Really Fin