elixir

@elixirlang / elixir-lang.org

The Plan

- Overview
- Elixir v1.3-dev
- Improving OTP
- R&D: GenStage & GenBroker

Overview

Elixir v1.0

- September/2014
- >180 contributors
- 3 books out
- First Elixirconf!

Elixir v1.1

- September/2015
- >295 contributors
- ~1000 packages on hex.pm
- ElixirConf US and ElixirConf EU

Elixir v1.2

- January/2016
- >384 contributors
- Since then...

FRIDAY, MARCH 4, 2016

St. Augustine, Florida

Building Community With Top Talks & Helpful Hacking

Register Today!

EMP.EX

Crevalle proudly presents

Empire City Elixir Conference 2016

A one-day conference for curious programmers

Saturday, May 21st

New York City

BUY TICKETS

About

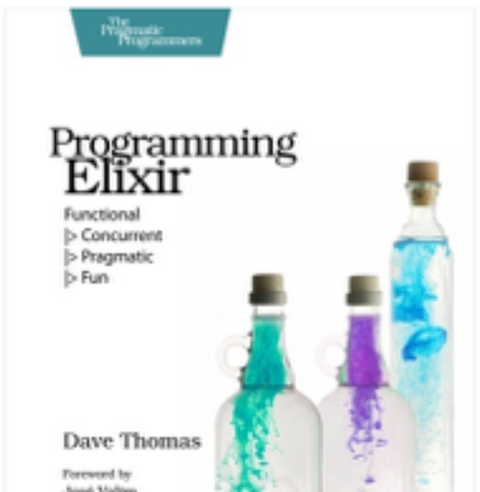
Learning resources

Our website provides a Getting Started guide to learn more about Elixir's foundation and explore how to build projects with Mix and OTP.

The Elixir Community has also produced plenty of resources to explore Elixir from different backgrounds and other perspectives. We are sure you will find a resource that follows your pace and interests.

Learn Elixir

Programming Elixir 1.2



You want to explore functional programming, but are put off by the academic feel (tell me about monads just one more time). You know you need concurrent applications, but also know these are almost impossible to get right. Meet Elixir, a functional, concurrent language built on the rock-solid Erlang VM.

Elixir's pragmatic syntax and built-in support for metaprogramming will make you productive and keep you interested for the long haul. And Programming Elixir is the

News: Elixir v1.2 released

Search...

JOIN THE COMMUNITY

- #elixir-lang on freenode IRC
- Elixir on Slack
- Elixir Forum
- elixir-talk mailing list
- @elixirlang on Twitter
- Meetups around the world
- Wiki with events, resources and talks organized by the community

IMPORTANT LINKS

- Source Code
- Issue tracker
- elixir-core mailing list (development)
- Crash course for Erlang developers

CODE EDITOR SUPPORT

- Emacs Mode
- Alchemist (Emacs Elixir Tooling)
- TextMate / Sublime Text Bundle

Elixir School

Lessons about the Elixir programming language, inspired by Twitter's Scala School

Available in Việt ngữ, 汉语, Español, 日本語, Português and Bahasa Melayu

Your feedback and participation is encouraged!

About Elixir

"Elixir is a dynamic, functional language designed for building scalable and maintainable applications." — elixir-lang.org

Elixir leverages the battle tested ErlangVM to build distributed and fault-tolerant systems with low-latency out of the box.

Features:

- Scalable
- Fault-tolerant
- Functional Programming
- Extensible

Share This Page

elixir forum

Sign Up | Log In

Welcome to the brand new Elixir Forum!

There's no time like the present to jump into Elixir - the functional language that's taking the world by storm 😍

As well as catering to the community's more general needs, we have a strong focus on learning, so if you've been curious about Elixir or are just starting out - join up - you'll be in great company!

This forum is also driven by the community. The more you participate here, the more trust you'll earn and the more mod-type tools you'll unlock - our way to reward regular members and your way to help give back to the community.

That's not all, we have lots of other features... so what are you waiting for? Take a sip of Elixir, sign up and join in the fun 💜

All Categories ▸ Latest Categories Top Stickies 2016 Giveaway!

Topic	Category	Users	Replies	Views	Activity
Who is currently developing with Elixir at work?	Elixir Chat		33	1.1k	5h
Elixir Blog Posts	Elixir Chat		41	1.1k	6h
Elixir Koans	Learning Elixir		2	113	8h
The Little Elixir & OTP Guidebook	Learning Elixir		13	286	8h
Blockchain	Elixir Chat		0	56	11h
Elixir regular expression editor & tester	Elixir Chat		3	87	14h
Portland Meetup - 27 April 2016	Confs & Meet Ups		3	79	15h

Using with Elixir

Simply specify your Mix dependencies as two-item tuples like `{:ecto, "~> 0.1.0"}` and Elixir will ask if you want to install Hex if you haven't already. After installed, you can run `$ mix local` to see all available Hex tasks and `$ mix help TASK` for more information about a specific task.

Using with Erlang

Download rebar3, put it in your `PATH` and give it executable permissions. Now you can specify Hex dependencies in your rebar.config like `{deps, [hackney]}`.

1 991 packages available

8 832 package versions

191 522 downloads yesterday

999 702 downloads last 7 days

26 600 719 downloads all time

MOST DOWNLOADED

cowboy
1 038 622 downloads
Small, fast, modular HTTP server written in Erlang.
published 624 days ago

NEW PACKAGES

tarantool
Tarantool client for Elixir language
published 1 hour ago

RECENTLY UPDATED

esip
ProcessOne SIP server component in Erlang
1.0.3 published 14 minutes ago

Productive. Reliable. Fast.

A productive web framework that
does not compromise speed and maintainability.

Build APIs, HTML 5 apps & more
See our guides

HOW IS PHOENIX DIFFERENT?

Phoenix brings back the simplicity and joy in writing modern web applications by mixing tried and true technologies with a fresh breeze of functional ideas.

Get started with Phoenix

BUILDING THE NEW WEB

Create rich, interactive experiences across browsers, native mobile apps, and embedded devices with our real-time streaming technology called Channels.

Learn about channels

BATTLE-PROVEN TECHNOLOGY

Phoenix leverages the Erlang VM ability to handle millions of connections alongside Elixir's beautiful syntax and productive tooling for building fault-tolerant systems.

More about Elixir & the Erlang VM

nerves

craft bulletproof firmware in the stunningly productive **elixir** language

platform

Using a lean, custom cross-compiled linux, nerves boots directly to the battle hardened BEAM VM, starting your application in seconds.

framework

Most devices need to get on a network, get discovered, update firmware, and deal with I/O of various kinds. You're not on your own.

tooling

Cross-compliation can be a total drag. Our tools make it smooth as silk. Go from "mix new" to running code on your device in minutes.

Nerves is young, but already powers rock-solid shipping industrial products! Check us out if you are a hearty experimenter and interested in a new way of creating embedded systems.

Nerves is fully open source. Nerves uses Buildroot to provide Linux, so some portions of the platform are licensed under the GPL.

LATEST NEWS

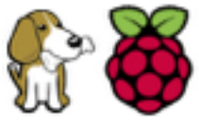
- Baking firmware on Mac OS X
- Examples you can build
- Intel Galileo now supported

JOIN THE COMMUNITY

- #nerves channel on Elixir Slack
- nerves-project mailing list
- Nerves on Github

Follow @NervesProject Tweet

HARDWARE SUPPORT



We support prototyping on Raspberry Pi, Pi 2, and Beaglebone Black.

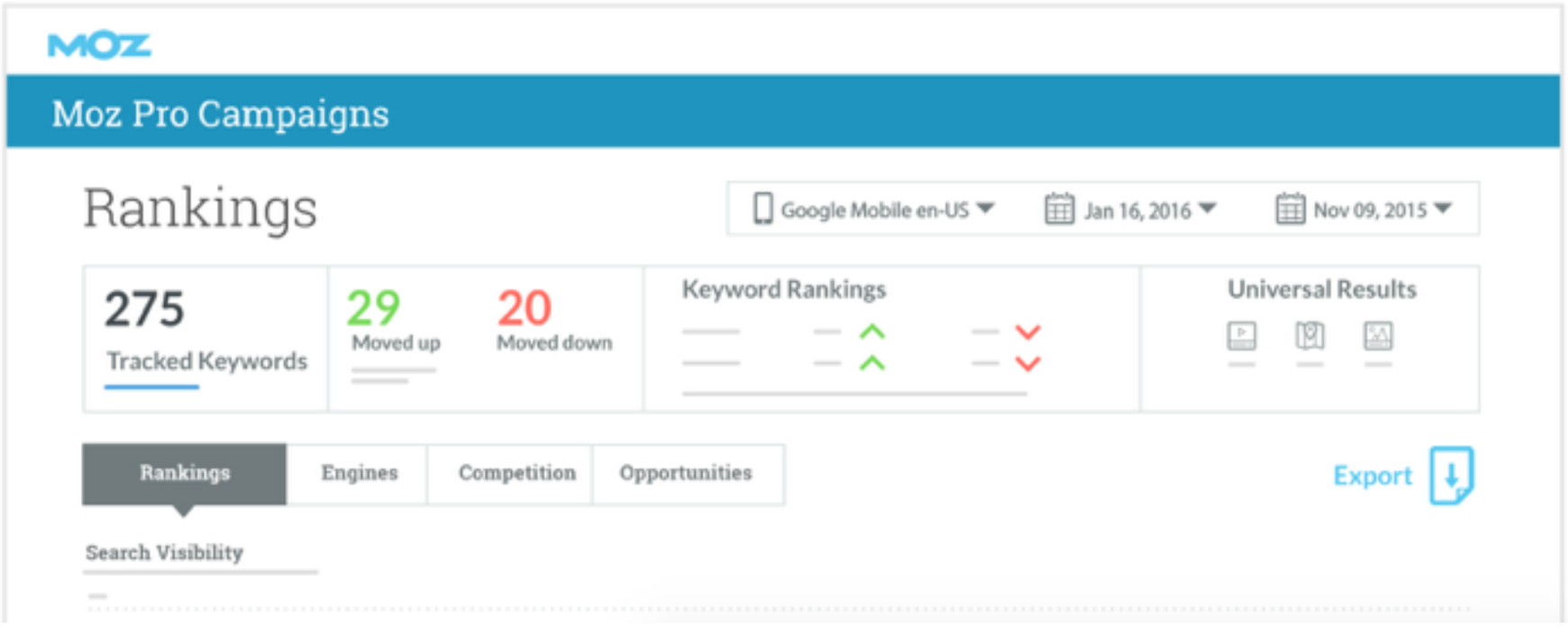
Moz Developer Blog

Unlocking New Features in Moz Pro with a Database-Free Architecture

Date / feb 16, 2016 / **Posted by** / Jeremy Modjeska / **Category** / Architecture, Development

Moz Pro is undertaking a comprehensive overhaul of our backend architecture to improve the performance and speed of our application and to unlock significant new features in high demand by our customers. We are abandoning MySQL database storage —our current infrastructure's decisive bottleneck—in favor of a database-free architecture and an Elixir-driven data indexing model. In this post, I'll discuss our new architecture, the competitive and technical reasons we chose to invest in this project, and some interesting implementation challenges we faced.

Overview of Moz Pro



Welcome to our dev blog!

Moz engineers writing about the work we do and the tech we care about.

Categories

Agile
API
Architecture
AWS
Caching
Cloud Computing
Communciation
conferences
Data Science
Dev Culture
Development
Launches
Linkscape
Operations
People Management
Post Mortems
Python
Rails
Random
Ruby
Scaling
Software Development Process
Uncategorized
White Papers

Introducing new open-source tools for the Elixir community

Steve Cohen

Steve Cohen is a heavy consumer of Elixir on the Product Platform team

The eighties aren't typically remembered as a somber and serious decade. Yet, while the masses were listening to Wang Chung's "Everybody have fun tonight" and hoping for a DeLorean based time machine, several engineers at a Swedish telecom company were accidentally inventing the future. They were solving a problem for telecommunications, but by a strange coincidence, the problem they were solving also turned out to be very similar to the problems faced by engineers that would see that DeLorean time machine after it completed its 30 year jump to the future.

Despite its many advantages, this magical system never really got traction, but spread like an urban legend: Spoken in hushed tones by seasoned engineers, who always heard about it third hand. Sure, you knew a cousin who had a roommate that knew a gal that ran a couple million connections off of two servers, but you never actually believed that stuff anyway.

Well, I'm here to tell you that the stories were real. And there's a sequel.

I am, of course, talking about Erlang, and while Erlang has had some recent

Elixir v1.3-dev

Calendar types

```
# Persist yesterday + 1 hour with Ecto
GoodTimes.Date.yesterday
|> Calendar.DateTime.from_erl
|> Calendar.DateTime.add(hours: 1)
|> Calendar.DateTime.to_erl
|> Ecto.DateTime.from_erl
```

Calendar types

- Time
- Date
- NaiveDateTime
- DateTime

Calendar types

- Common ground for interoperability
- Pending: basic conversion functions
- Support multiple calendars (ISO8601)

Calendar types

- Lau Taarnskov (Calendar)
- Paul Schoenfelder (Timex)

with (recap)

```
with :ok <- validate_strategy(strategy),
     :ok <- validate_restarts(max_restarts),
     :ok <- validate_seconds(max_seconds) do
  {:ok, %DynamicSupervisor{...}}
end
```

with (recap)

```
defp validate_restarts(restart) when is_integer(restart),
  do: :ok
defp validate_restarts(_),
  do: {:error, "max_restarts must be an integer"}

defp validate_seconds(seconds) when is_integer(seconds),
  do: :ok
defp validate_seconds(_),
  do: {:error, "max_seconds must be an integer"}
```

with (recap)

```
with :ok <- validate_strategy(strategy),
     :ok <- validate_restarts(max_restarts),
     :ok <- validate_seconds(max_seconds) do
  {:ok, state}
end
# {:ok, state} OR {:error, message}
```

with (recap)

```
res =
  with :ok <- validate_strategy(strategy),
       :ok <- validate_restarts(max_restarts),
       :ok <- validate_seconds(max_seconds) do
    {:ok, state}
  end

case res do
  {:ok, state} ->
    {:ok, state}
  {:error, message} ->
    {:error, String.upcase(message)}
end
```

with-else

```
with :ok <- validate_strategy(strategy),
     :ok <- validate_restarts(max_restarts),
     :ok <- validate_seconds(max_seconds) do
  {:ok, state}
else
  {:error, message} ->
    {:error, String.upcase(message)}
end
```

ExUnit

```
10) test maps; mixed diff (Difference)
    lib/ex_unit/examples/difference.exs:58
    Assertion with == failed
    code: map1 == map2
    lhs:  %{11 => 11, 39 => 39, 34 => 34, 26 => 26, 15 => 15, 20 => 20, 17 => 17, 25 => 25,
            13 => 13, 8 => 8, 36 => 36, 7 => 7, 1 => 1, 32 => 32, 37 => 37, 35 => 35, 3 => 3,
            6 => 6, 2 => 2, 10 => 10, 9 => 9, 19 => 19, 14 => 14, 5 => 5, 18 => 18, 31 => 31,
            22 => 22, 29 => 29, 21 => 21, 27 => 27, 24 => 24, 40 => 40, 30 => 30, 23 => 23,
            28 => 28, 16 => 16, 38 => 38, 4 => 4, 12 => 12}
    rhs:  %{11 => 11, 39 => 39, 34 => 34, 26 => 26, 15 => 15, 20 => 20, 17 => 17, 25 => 25,
            13 => 13, 36 => 36, 1 => 1, 32 => 32, 37 => 37, 35 => 35, 3 => 3, 2 => 2, 33 => 33,
            19 => 19, 14 => 14, 18 => 18, 31 => 31, 22 => 22, 29 => 29, 21 => 21, 27 => 27,
            24 => 24, 40 => 40, 30 => 30, 23 => 32, 28 => 28, 16 => 16, 38 => 38, 4 => 4,
            12 => 12}
    diff: %{23 => 232 (off by +9), 8 => 8, 7 => 7, 6 => 6, 10 => 10, 9 => 9, 5 => 5, 33 => 33, ...}
    stacktrace:
      lib/ex_unit/examples/difference.exs:61
```

Mix Archives (recap)

```
$ mix archive.install
  phoenixframework.org/phoenix.ez

$ mix phoenix.new new_app
```

Mix Escripts

```
$ mix escript.install
  phoenixframework.org/phoenix

$ phoenix new_app
```

Mix Escripts

- Added as an alternative to archives
- Run apart from Mix
- Do not depend on the Elixir version installed locally

MIX_DEBUG=1

```
~/OSS/mime[master]$ MIX_DEBUG=1 mix compile
** Running mix loadconfig (inside MIME.Mixfile)
** Running mix compile (inside MIME.Mixfile)
** Running mix loadpaths (inside MIME.Mixfile)
** Running mix deps.check (inside MIME.Mixfile)
** Running mix archive.check (inside MIME.Mixfile)
** Running mix compile.all (inside MIME.Mixfile)
** Running mix compile.yecc (inside MIME.Mixfile)
** Running mix compile.leex (inside MIME.Mixfile)
** Running mix compile.erlang (inside MIME.Mixfile)
** Running mix compile.elixir (inside MIME.Mixfile)
Compiled lib/mime.ex
** Running mix compile.app (inside MIME.Mixfile)
Generated mime app
** Running mix compile.protocols (inside MIME.Mixfile)
Consolidated List.Chars
Consolidated Collectable
Consolidated String.Chars
Consolidated Enumerable
Consolidated IEx.Info
```

mix *.tree

```
$ mix app.tree
$ mix deps.tree
```

mix *.tree

```
~/OSS/phoenix/installer/demo[master *%]$ mix deps.tree
demo
├── gettext ~> 0.11 (Hex package)
├── phoenix_pubsub (https://github.com/phoenixframework/phoenix_pubsub.git)
├── cowboy ~> 1.0 (Hex package)
|   ├── cowlib ~> 1.0.0 (Hex package)
|   └── ranch ~> 1.0 (Hex package)
├── phoenix_html ~> 2.5 (Hex package)
|   └── plug ~> 0.13 or ~> 1.0 (Hex package)
|       └── cowboy ~> 1.0 (Hex package)
├── phoenix (../..) *override*
|   ├── cowboy ~> 1.0 (Hex package)
|   ├── plug ~> 1.0 (Hex package)
|   |   └── cowboy ~> 1.0 (Hex package)
|   ├── phoenix_pubsub (https://github.com/phoenixframework/phoenix_pubsub.git)
|   └── poison ~> 1.5 or ~> 2.0 (Hex package)
├── phoenix_live_reload ~> 1.0 (Hex package)
|   ├── phoenix ~> 0.16 or ~> 1.0 (Hex package)
|   └── fs ~> 0.9.1 (Hex package)
├── postgrex >= 0.0.0 (Hex package)
```

Contributions?

```
$ mix app.tree --dot
$ mix deps.tree --dot
```

http://tuvistavie.com/2016/elixir-1-3/

Daniel Perez

software engineer interested in programming languages and functional programming

Blog

Photo and illustration by Ai Miyuki

21 Apr 2016 • on Elixir

What's coming in Elixir 1.3

I recently gave a talk about Elixir 1.3 in Tokyo, and spoke about the changes, new features, improvements and all the awesome stuff coming in Elixir 1.3. I decided to write this as a blog post, with a little more details, and some links for those who want to check in more details.

Here is a short table of contents:

- Deprecation of imperative assignment
- `with` on steroids
- Calendar datatypes
- `escript` installation related tasks
- ExUnit diff
- `make` compiler addition
- Changes in `defdelegate`
- `Process.sleep` addition
- Support of OTP `optionalcallback`

Improving OTP

Improving OTP

- Simple one for one supervisors
- GenEvent

Supervision strategies

- one_for_one
- one_for_all
- rest_for_all
- simple_one_for_one

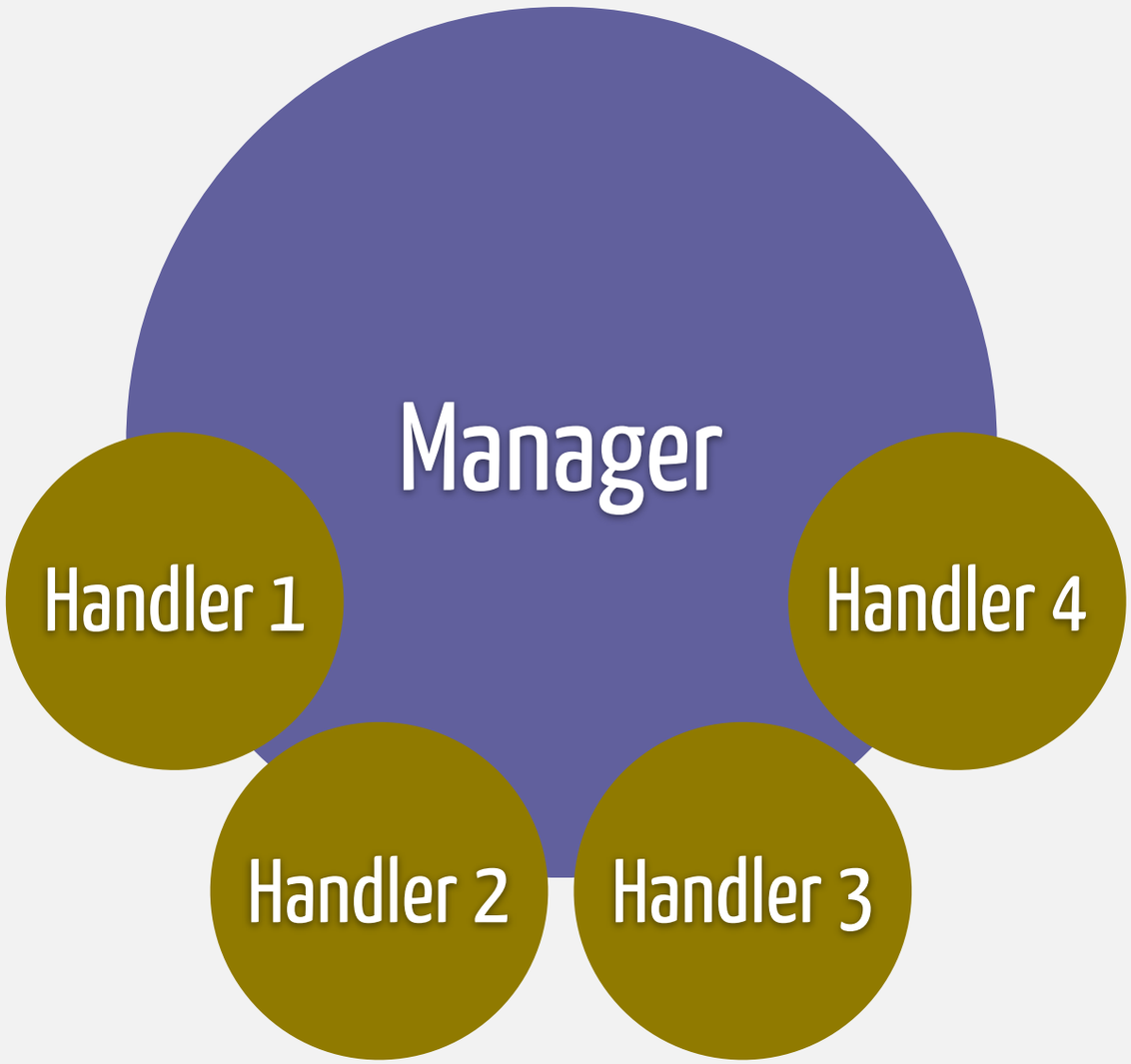
Simple one for one supervisor

- Used for spawning children dynamically
- init/1 must return a single child
- Many APIs expect different values
- etc

DynamicSupervisor

- Implements simple_one_for_one (simply called “one_for_one”)
- Push: limits max children?
- Pull: better back-pressure?

GenEvent



GenEvent

- Manager broadcasts events to handlers
- Handlers do not exploit concurrency
- Awkward error semantics:

  Handlers are removed and not re-added

Gen???

- Handlers are different processes
- Supports multiple “strategies"
- Provides back-pressure

R&D: GenStage & GenBroker

Collections

enumerable

```
widgets
|> Enum.filter(fn b -> b.color == :red end)
|> Enum.map(fn b -> {b.title, b.height} end)
|> Enum.into(%{})
```

collectable

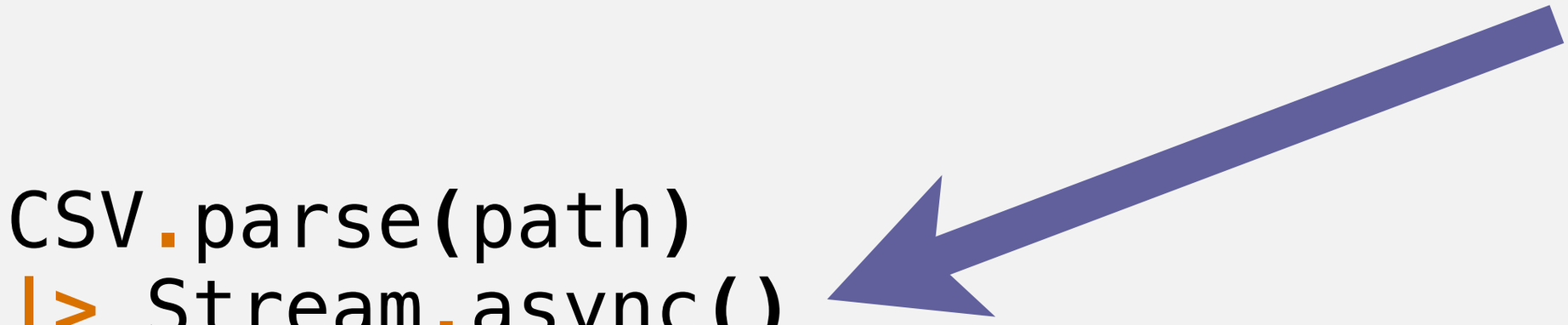
Collections

```
CSV.parse(path)
|> Enum.filter(fn b -> b.color == :red end)
|> Enum.map(fn b -> {b.title, b.height} end)
|> Enum.into(IO.stream(:stdio, :inspect))
```

Collections + Laziness

```
CSV.parse(path)
|> Stream.filter(fn b -> b.color == :red end)
|> Stream.map(fn b -> {b.title, b.height} end)
|> Stream.into(IO.stream(:stdio, :inspect))
|> Stream.run()
```

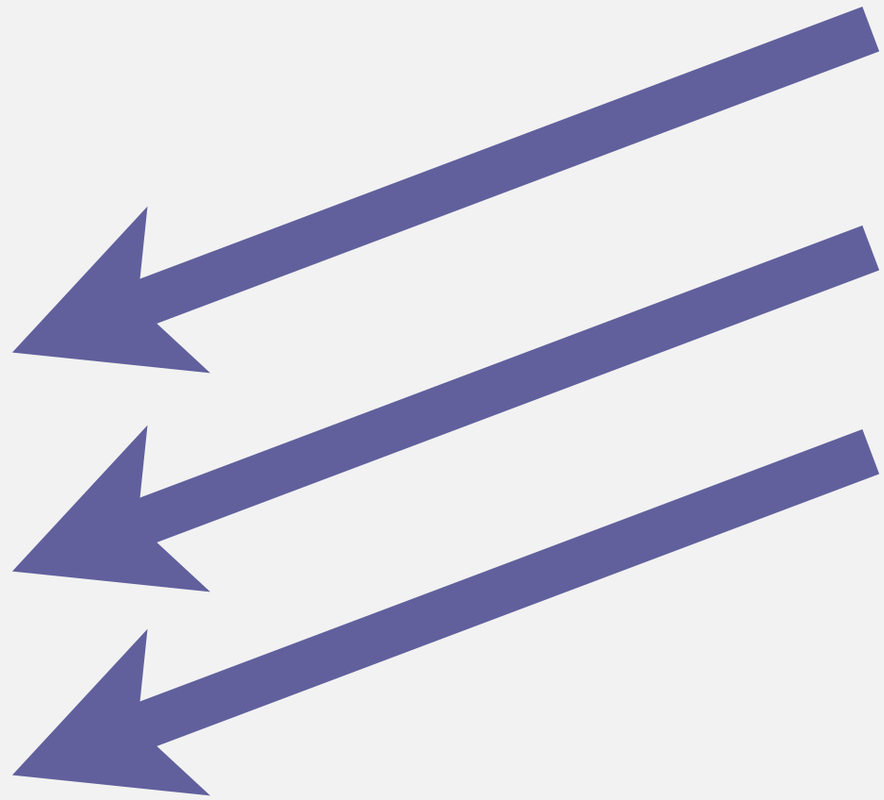
Pipeline parallelism



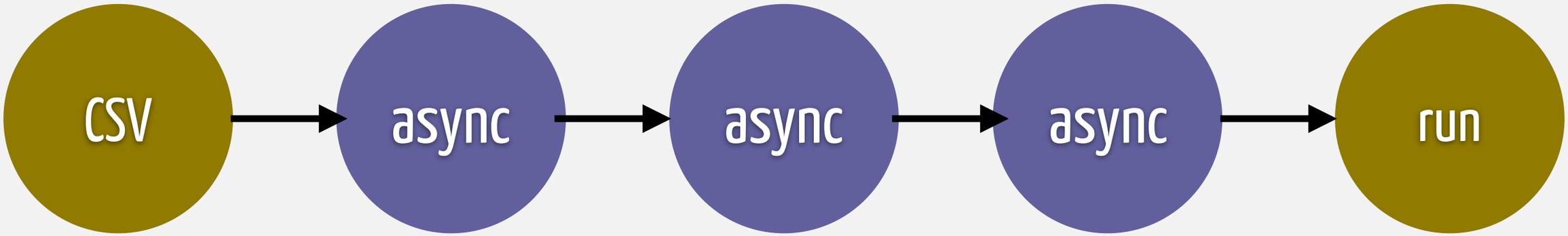
```
CSV.parse(path)
|> Stream.async()
|> Stream.filter(fn b -> b.color == :red end)
|> Stream.map(fn b -> {b.title, b.height} end)
|> Stream.into(IO.stream(:stdio, :inspect))
|> Stream.run()
```

Pipeline parallelism

```
CSV.parse(path)
|> ...
|> Stream.async()
|> ...
|> Stream.async()
|> ...
|> Stream.async()
|> ...
|> Stream.run()
```

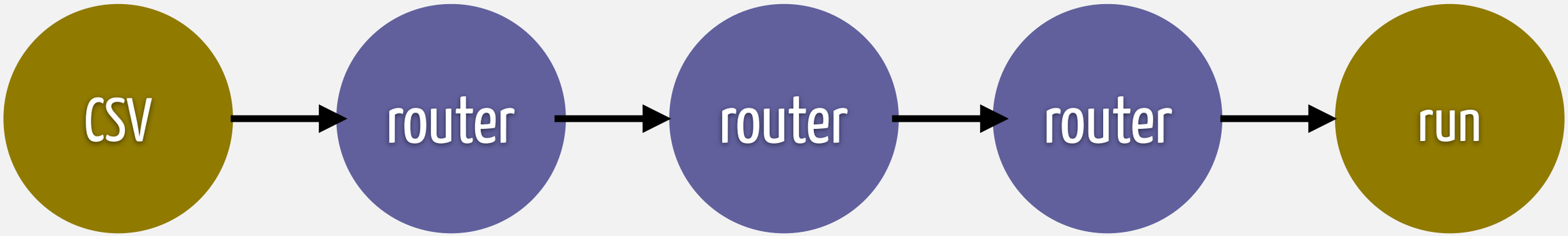


Pipeline parallelism

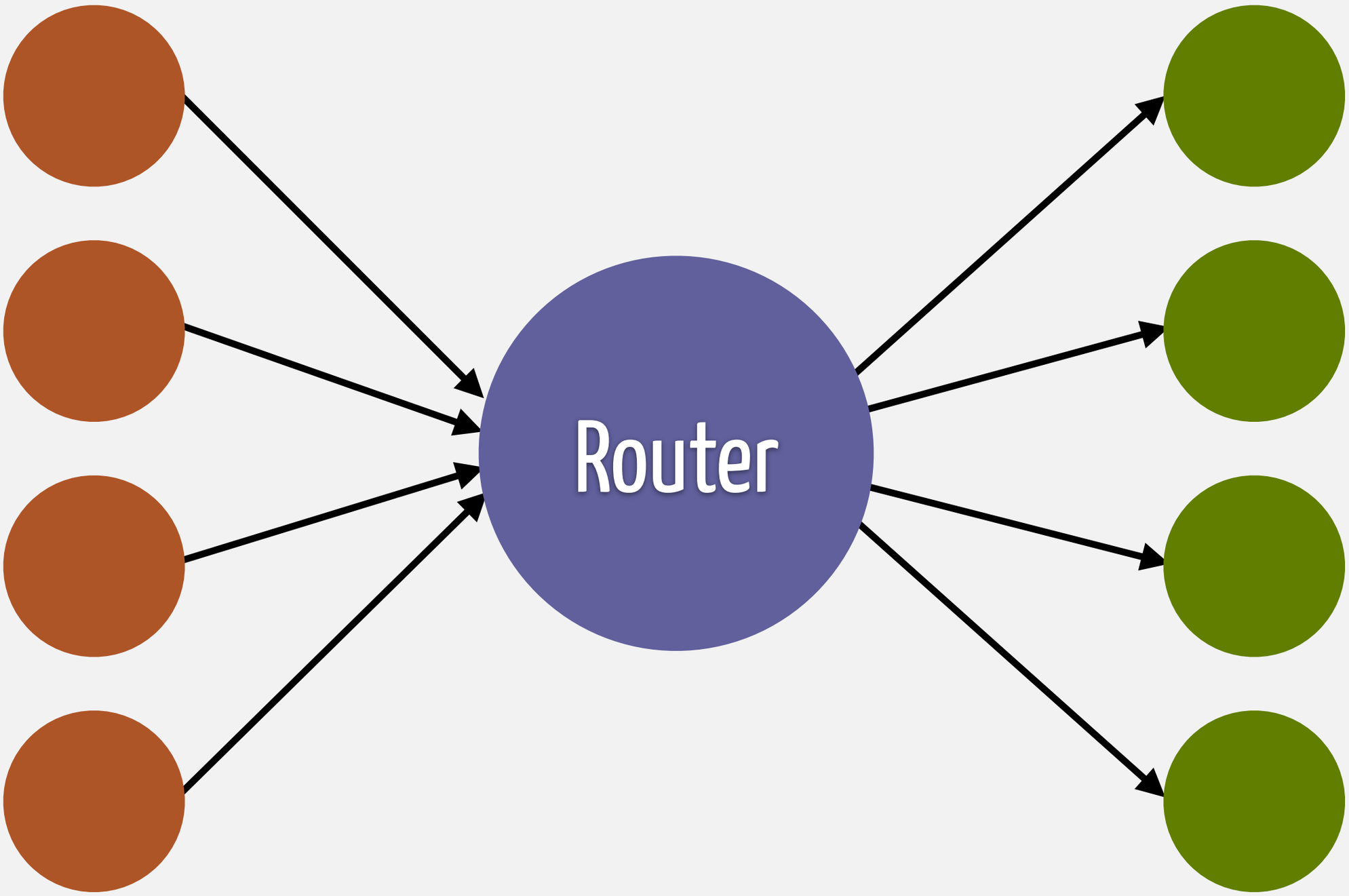


GenRouter

GenRouter



GenRouter

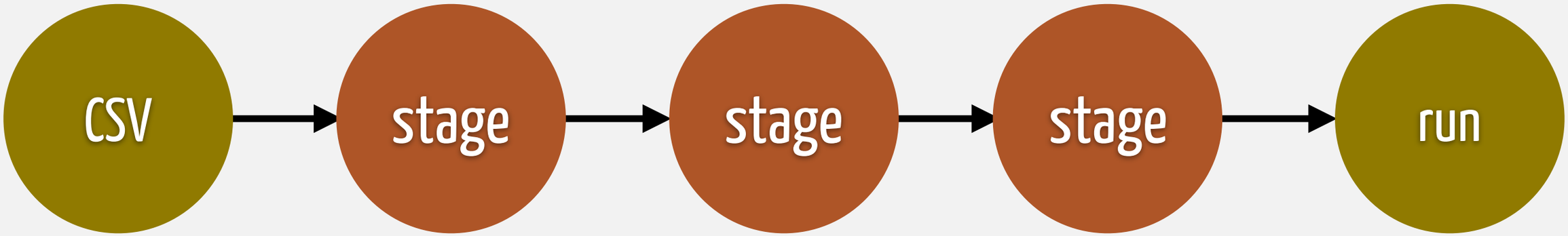


GenRouter

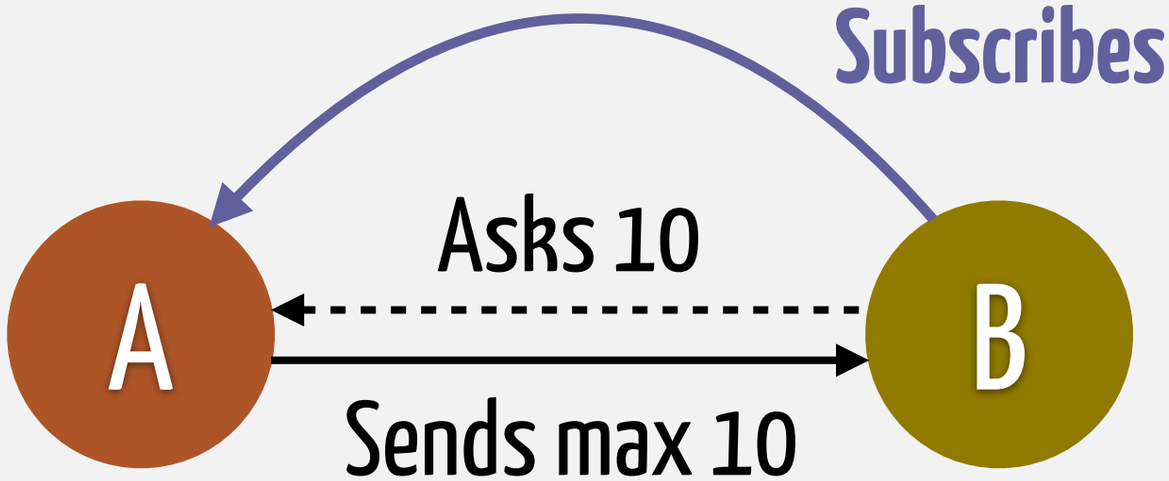
- Couples transformations with event balancing
- Event balancing incurs extra copying

GenStage & GenBroker

GenStage



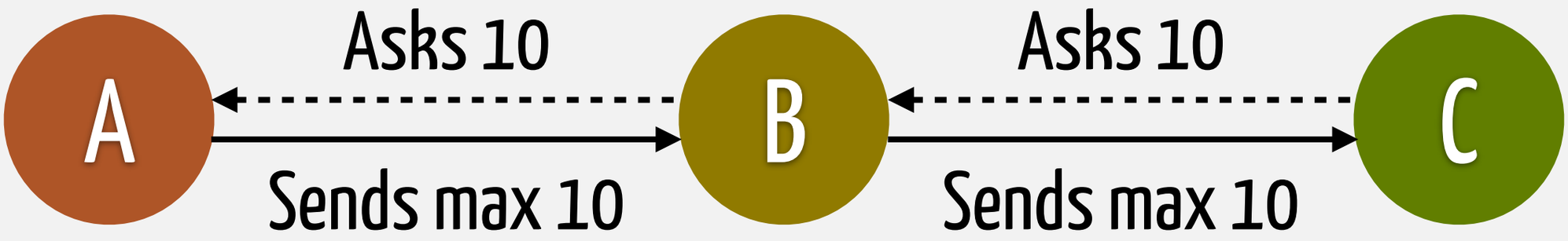
GenStage: Demand-driven



Producer Consumer

1. consumer subscribes to producer
2. consumer sends demand
3. producer sends events

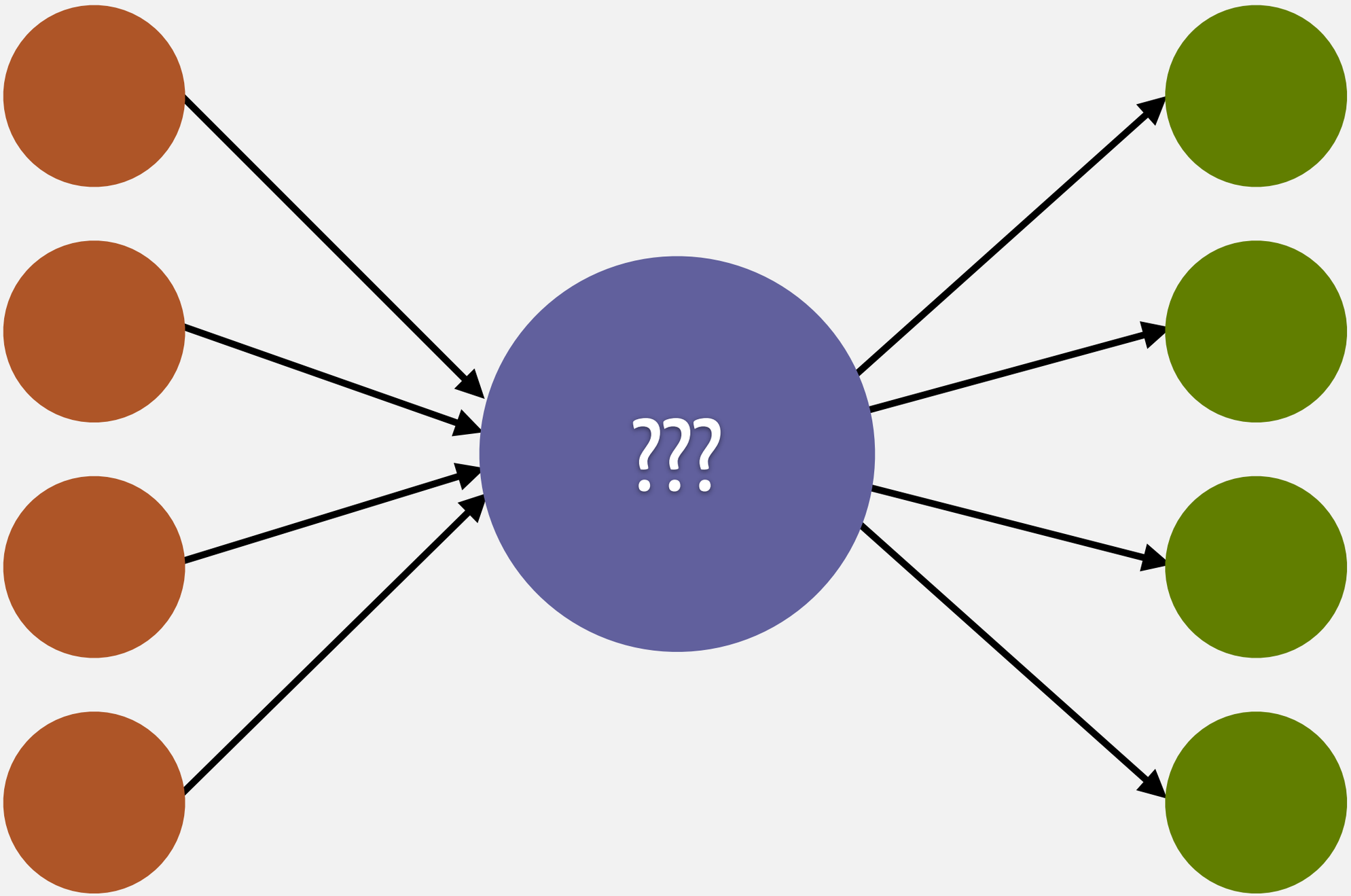
GenStage: Demand-driven



Demand-driven

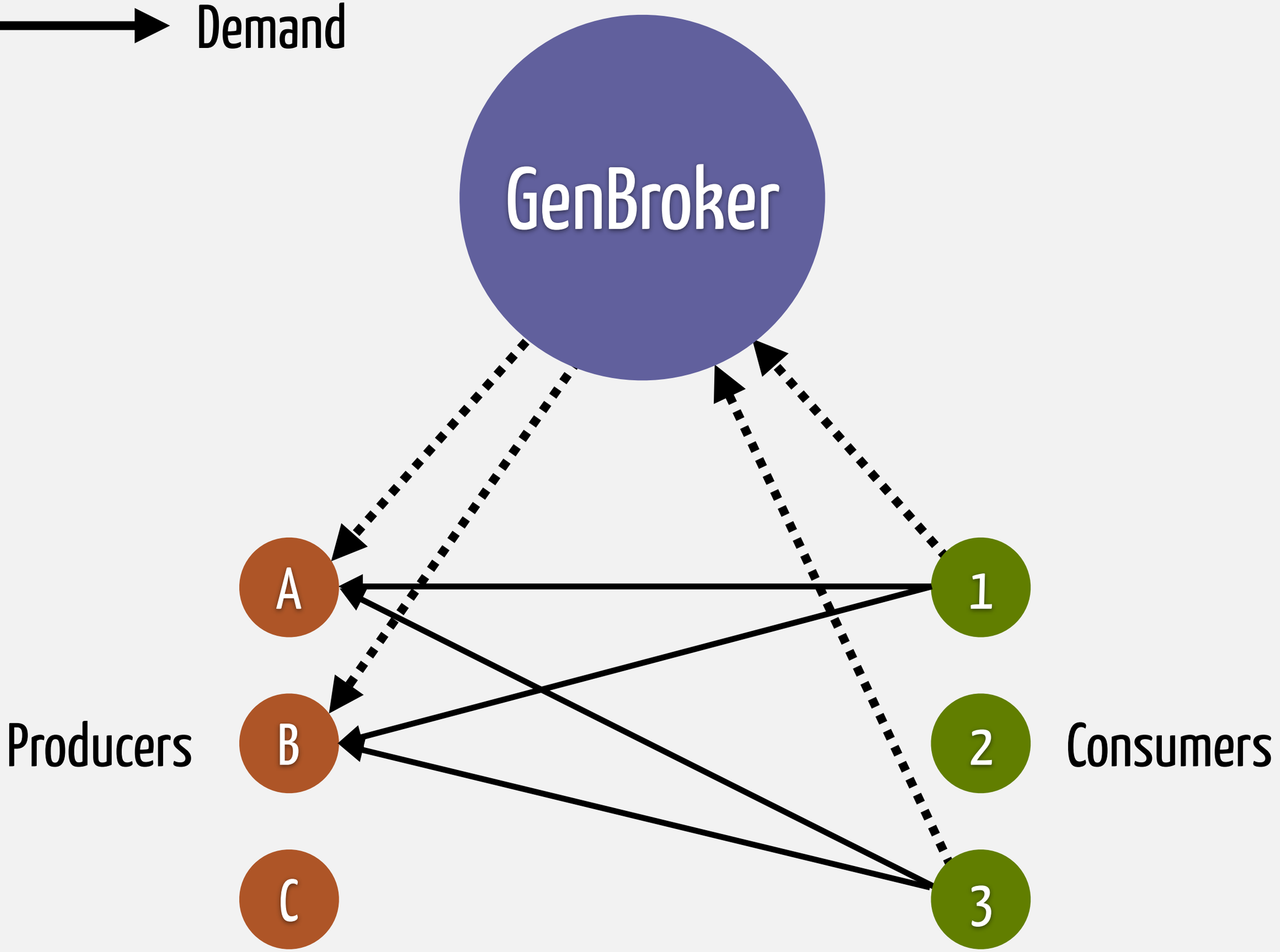
- It is a message contract
- It pushes back-pressure to the boundary
- GenStage is one impl of this contract
- Inspired by Akka Streams

?

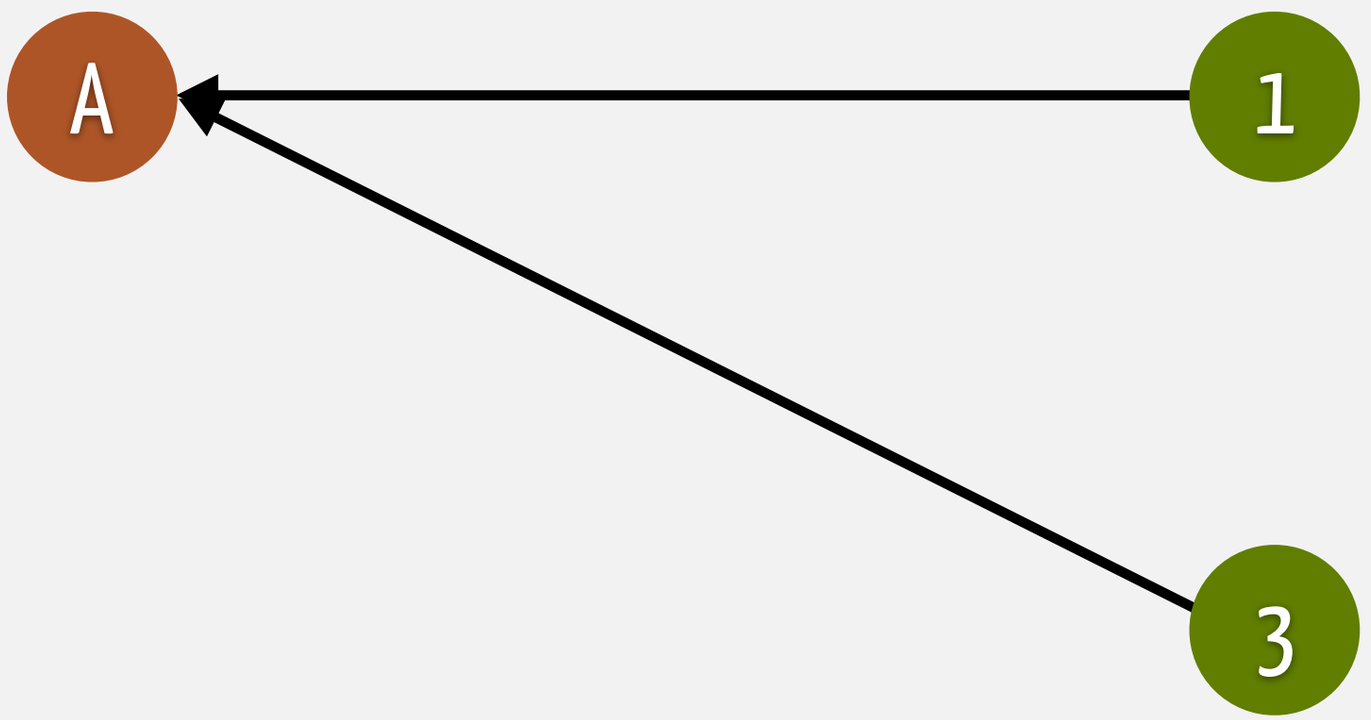


Subscribe

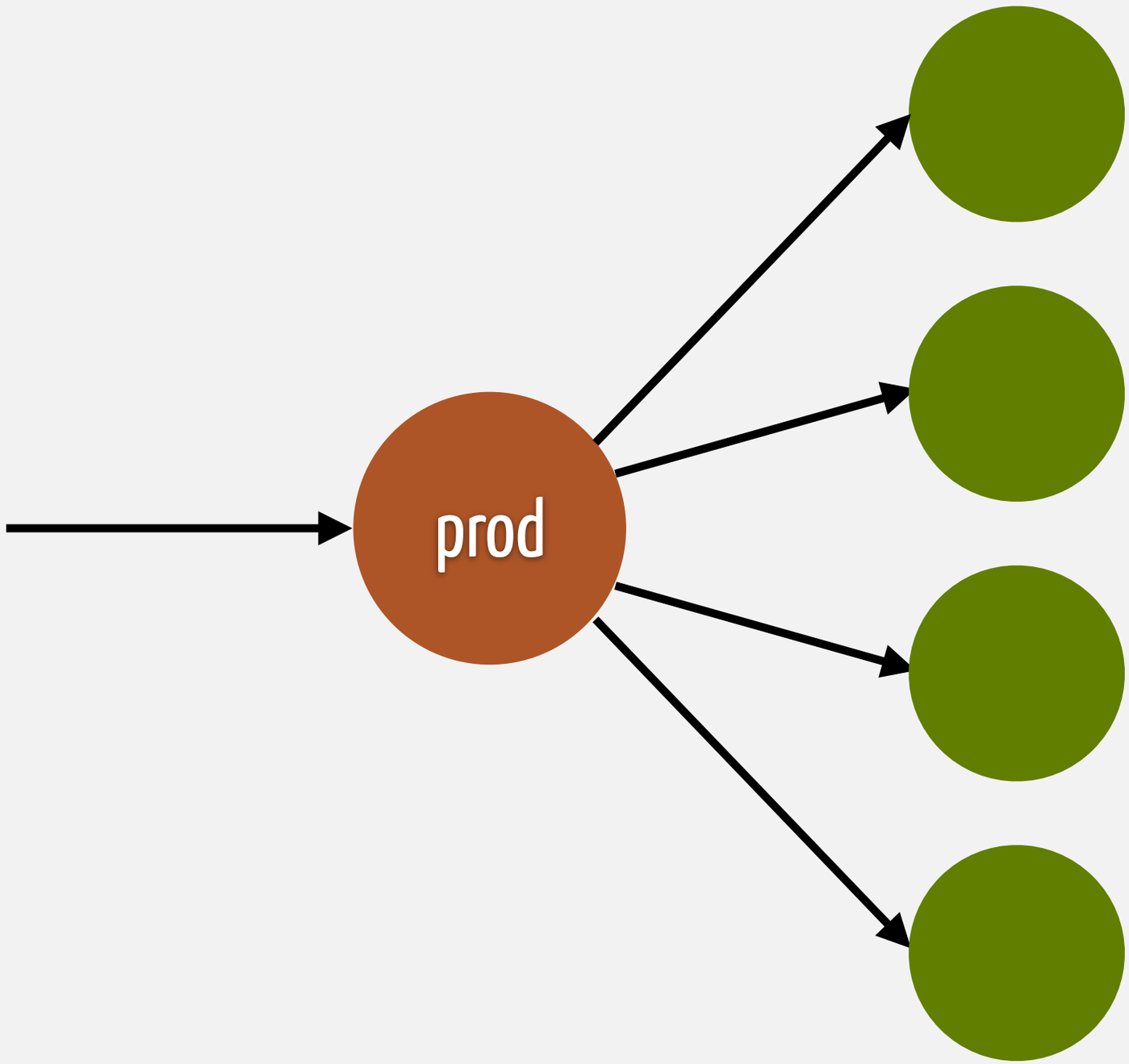
Demand



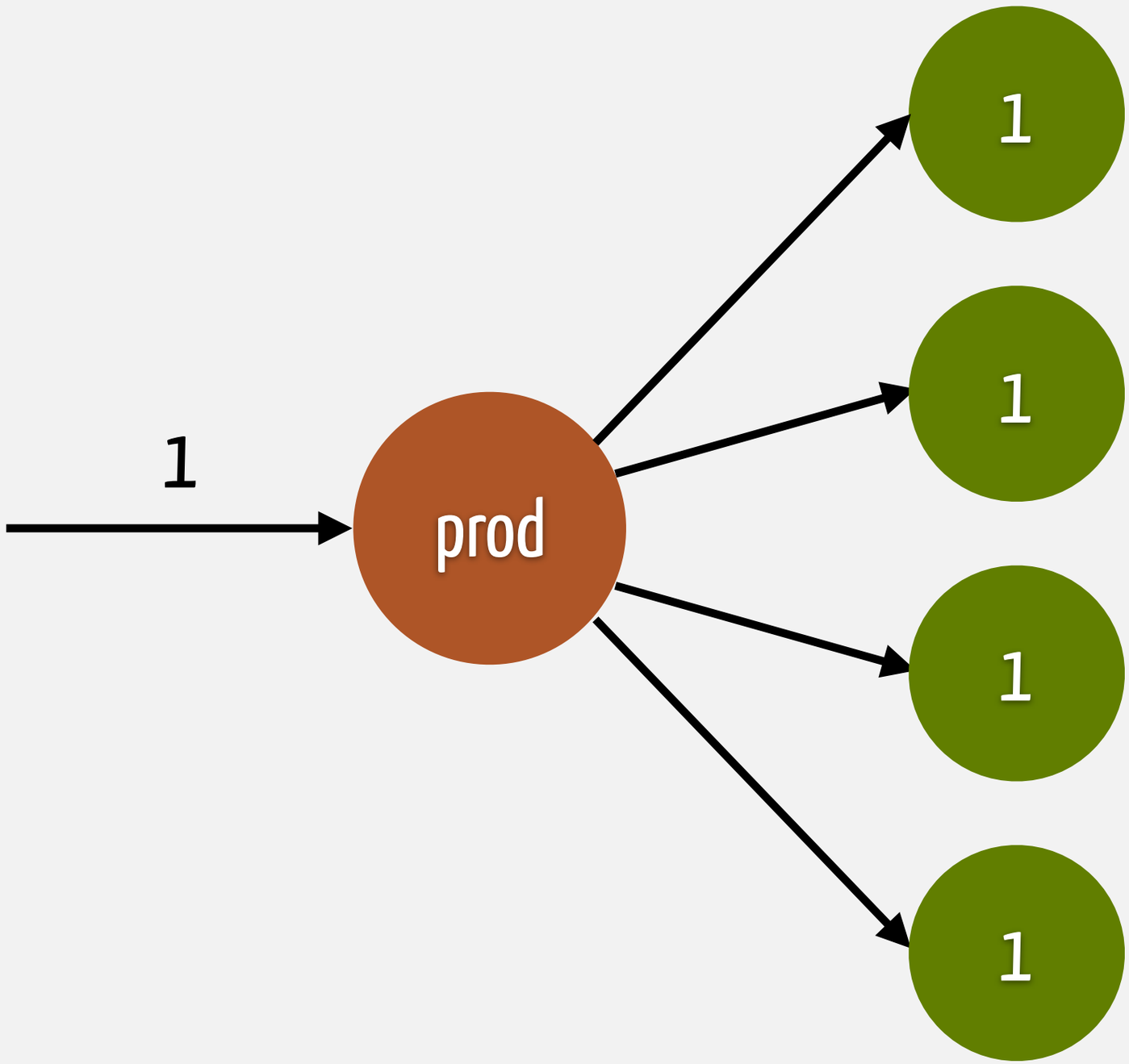
GenBroker



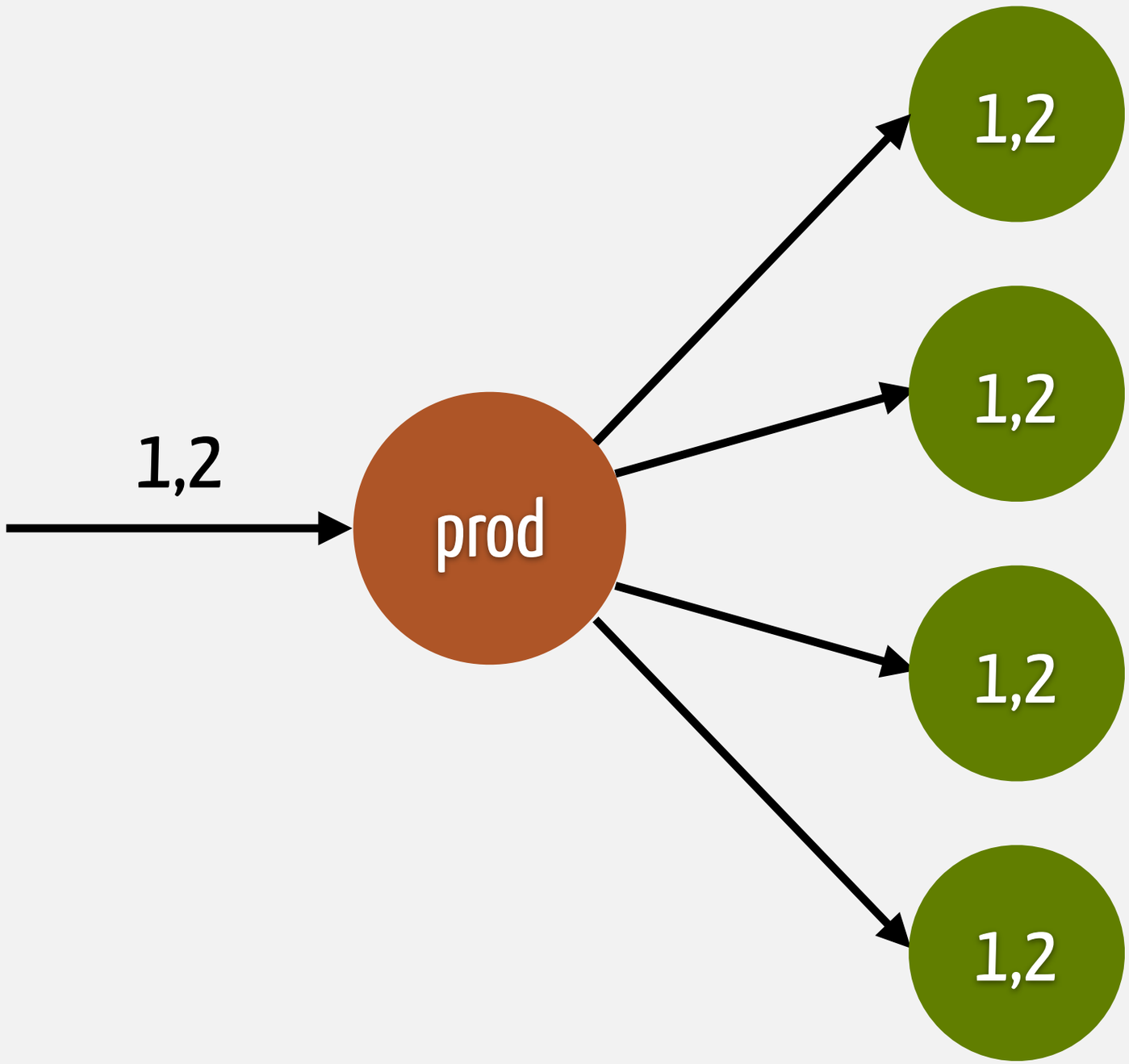
Broker strategy: broadcast



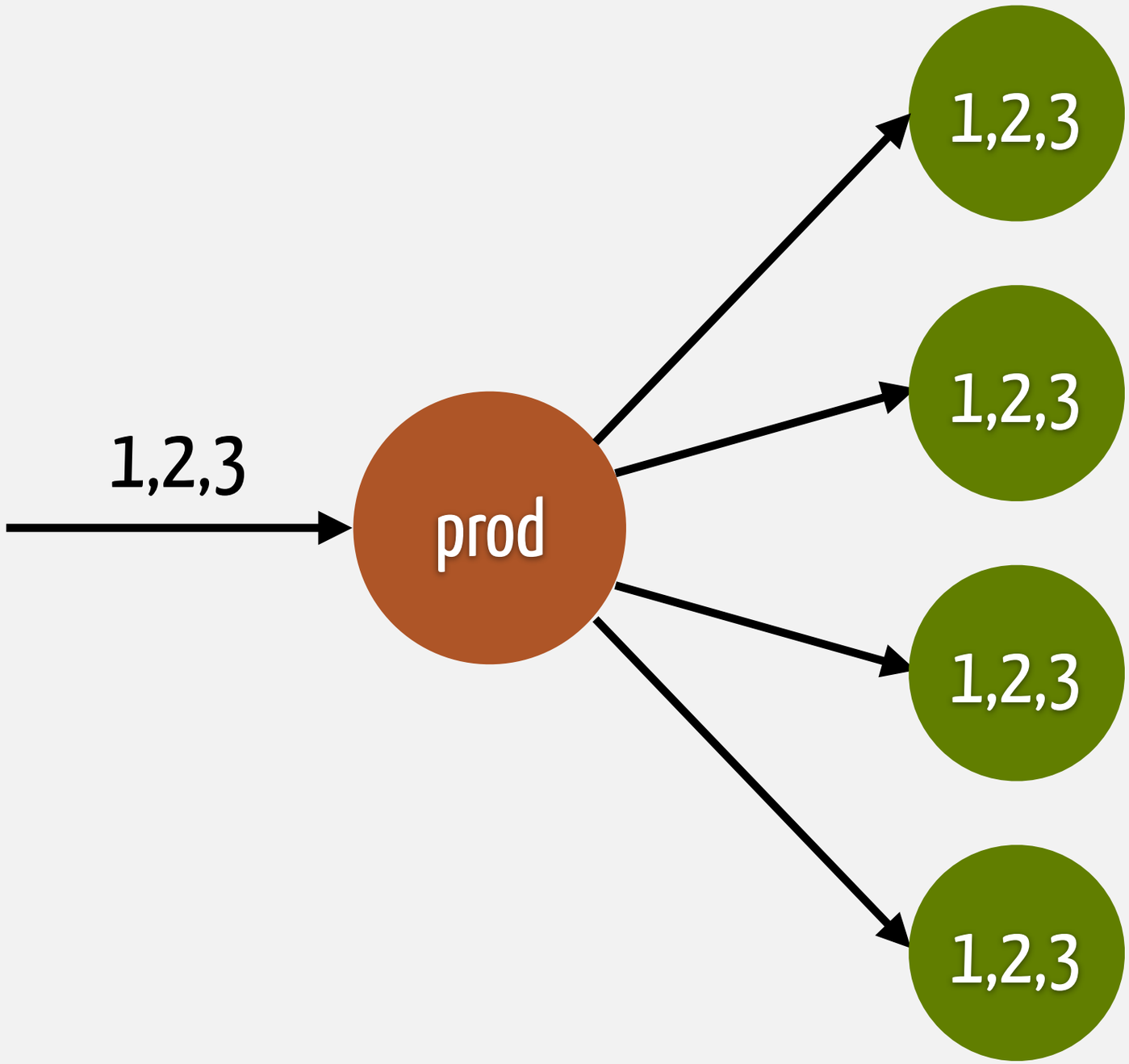
Broker strategy: broadcast



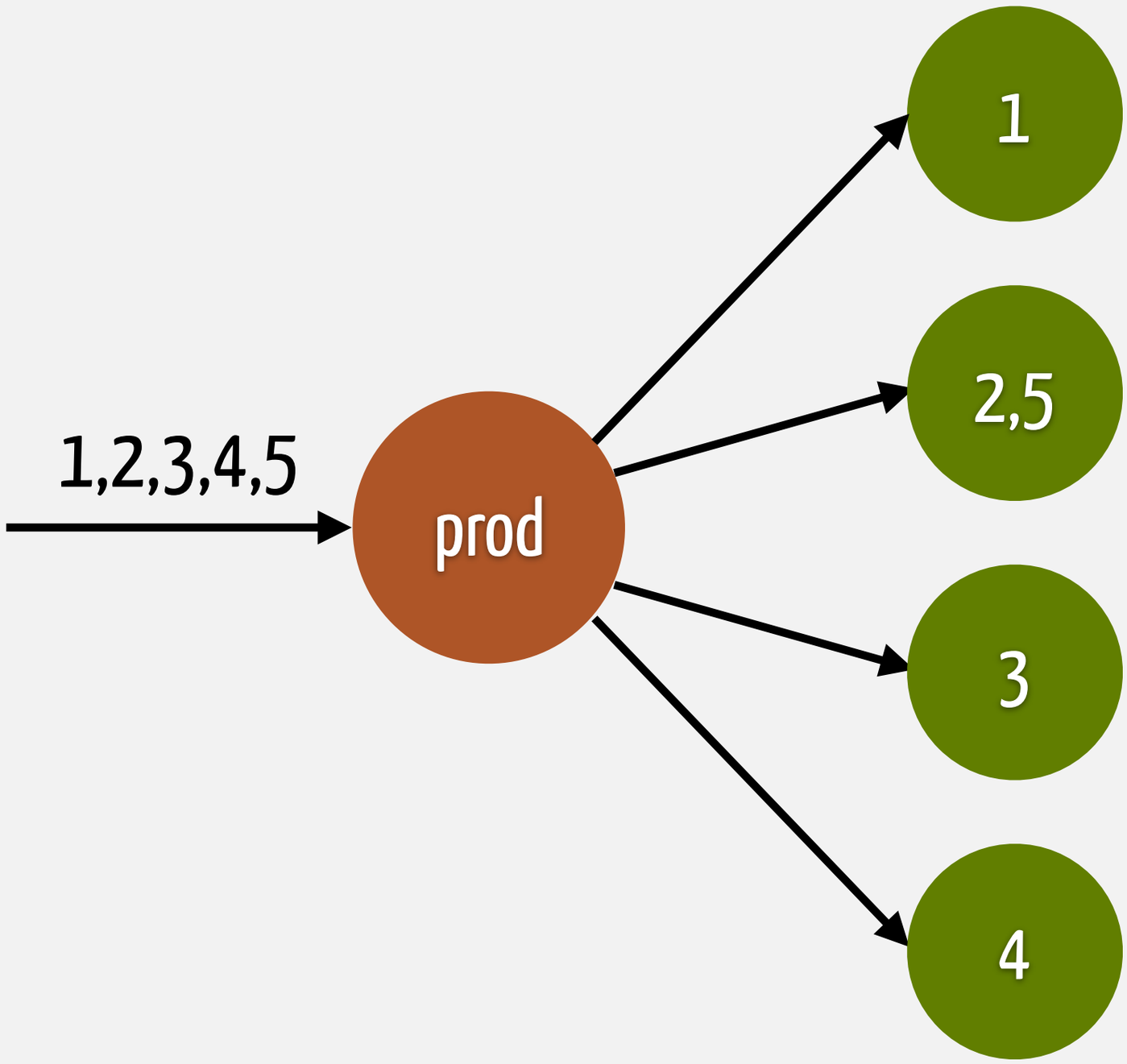
Broker strategy: broadcast



Broker strategy: broadcast



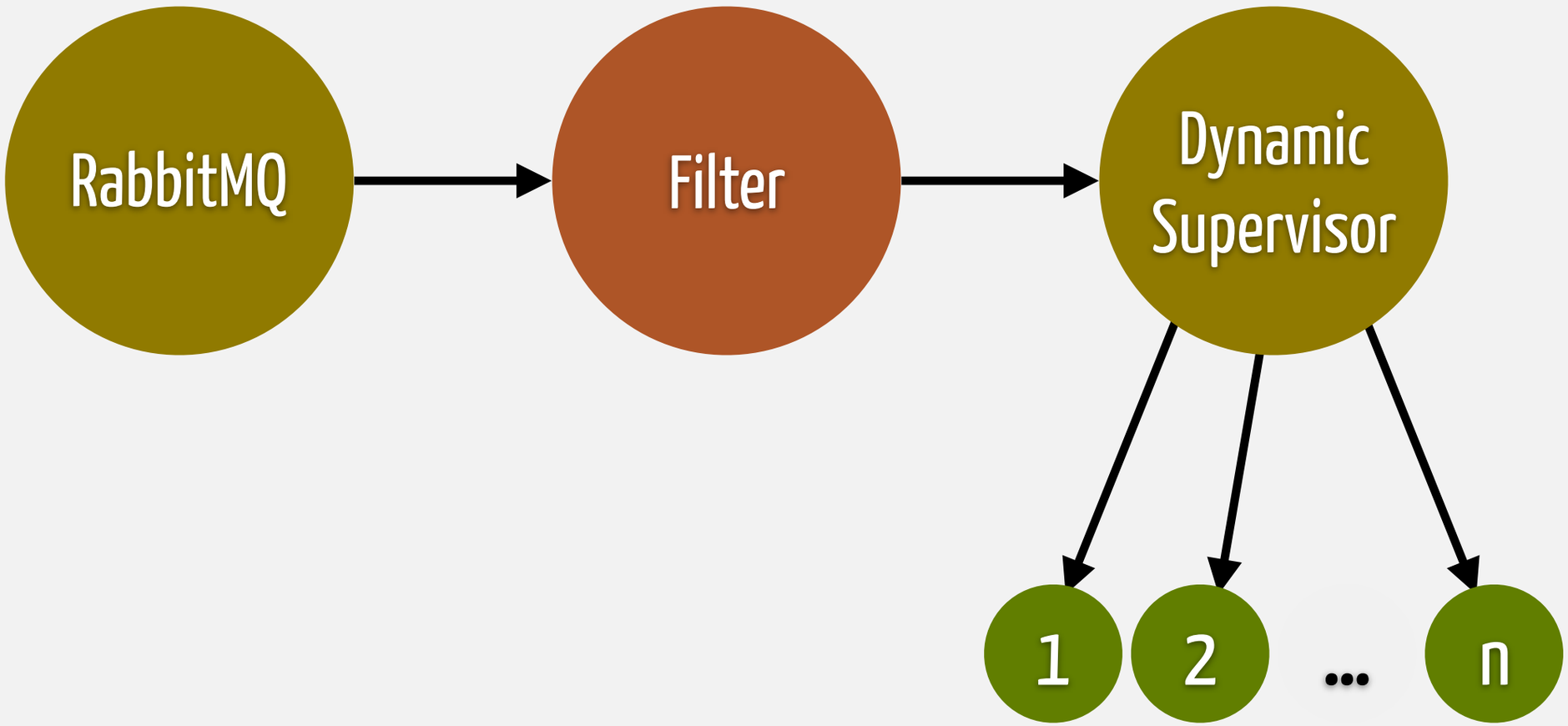
Broker strategy: demand size



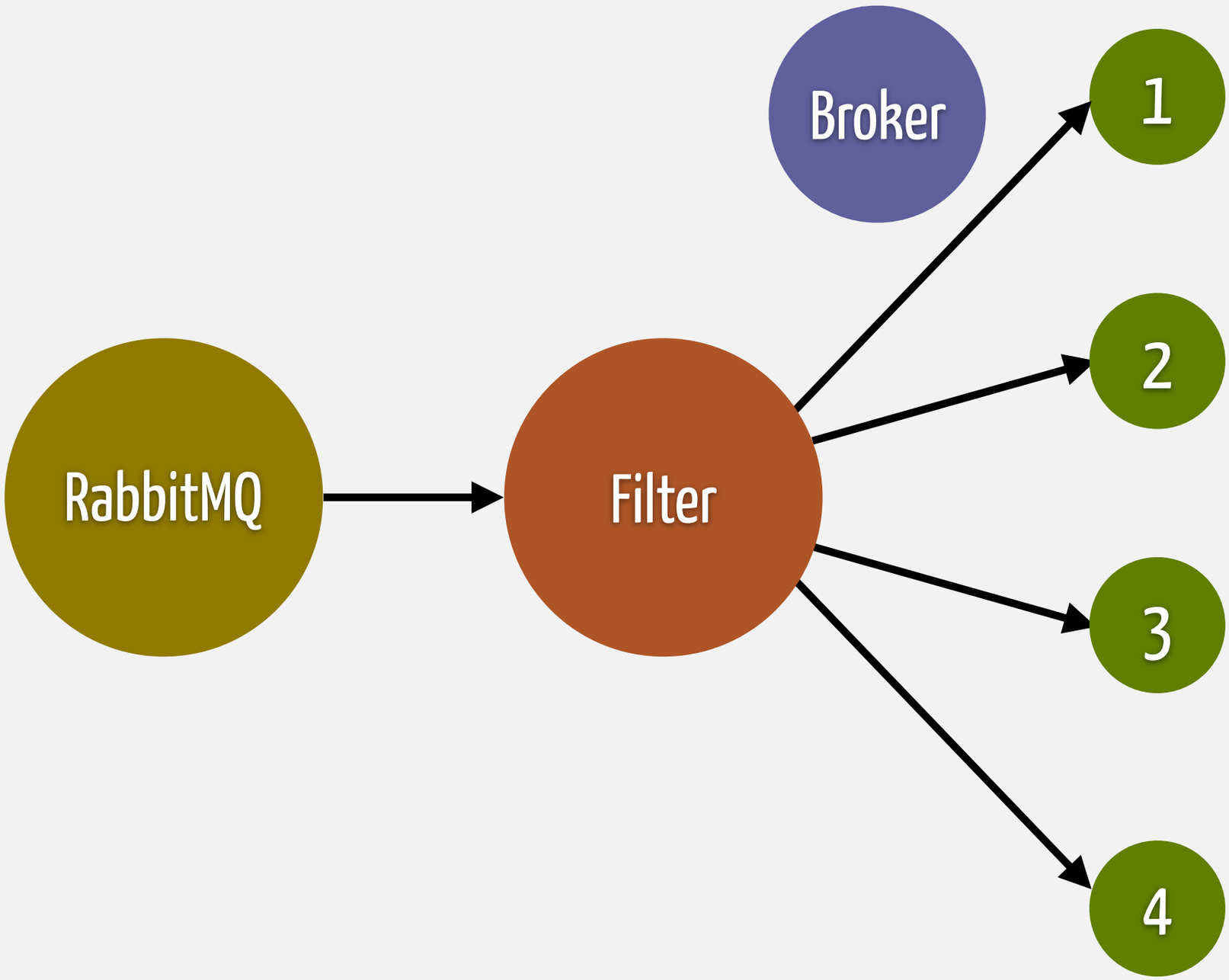
GenStage & GenBroker

- Solves the issues from GenRouter
- Stages are separate processes
- Provides back-pressure
- Supports multiple broker strategies

Examples



Examples



Improving OTP

- Demand-driven supervisors
  - Children are spawned on demand
- GenEvent -> GenBroker
  - Solves all issues we had with GenEvent

The path forward

- Finish GenStage
- Finish GenBroker
- Finish DynamicSupervisor
- https://github.com/elixir-lang/gen_broker

plataformatec

consulting and software engineering

elixir

@elixirlang / elixir-lang.org