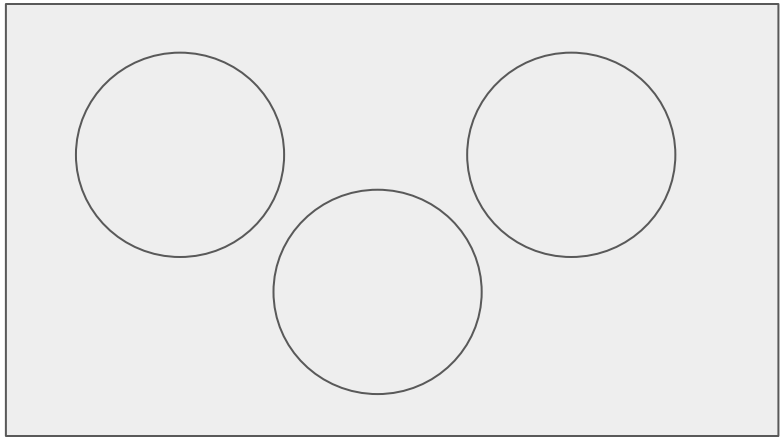
Why We Chose Elixir
And
How It Helped Tackle Concurrency and Scalability

Montreal Elixir, September 2017

by Ahmad Ferdous Bin Alam

Topics

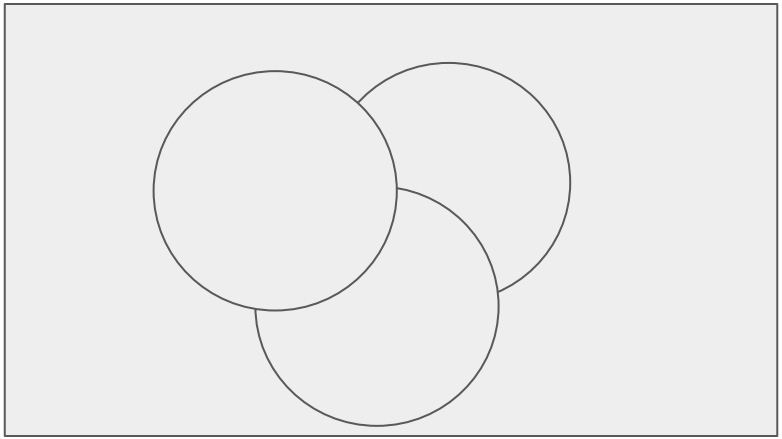
- What problem we are trying to solve
- What the requirements are
- Why Node.js was not a good fit for the problem
- How Elixir came to the rescue
- How we tackled the challenges with the help of Elixir and OTP
- Things to tackle in near future



Till recently

- Multiple subsystems
- Multiple 3rd party service providers.
- No service is consumed by more than one subsystem.

And then



- Some services are consumed by more than one subsystem!

Problem

- Multiple subsystems consuming the same third-party APIs from multiple providers
  - Rate limit not properly enforced
  - Duplicated effort

- Solution: A central API gateway

Requirements

- ❑ Enforce rate limit

- ❑ Common or global rate
- ❑ Client-specific rate
- ❑ API endpoint-specific rate

- ❑ Burst rate
- ❑ Quota limit
- ❑ Concurrent request limit

Requirements

- Enforce rate limit

10,000 updates per day
250,000 reads per day

5 queries per second (QPS)

5 concurrent requests max

Requirements

- ❑ Enforce rate limit
- ❑ Robust error handling

Requirements

- ❑ Enforce rate limit
- ❑ Robust error handling

- ❑ Fault tolerance
  - ❑ Persistence (consumed limit, in-progress requests, backlog requests)

Requirements

- ❑ Enforce rate limit
- ❑ Robust error handling

- ❑ Fault tolerance
- ❑ Extensibility

Requirements

- ❏ Enforce rate limit
- ❏ Robust error handling

- ❏ Fault tolerance
- ❏ Extensibility
- ❏ Scalability

Requirements

- ❑ Enforce rate limit
- ❑ Robust error handling

- ❑ Fault tolerance
- ❑ Extensibility
- ❑ Scalability
- ❑ Visibility

Requirements

- ❑ Enforce rate limit
- ❑ Robust error handling

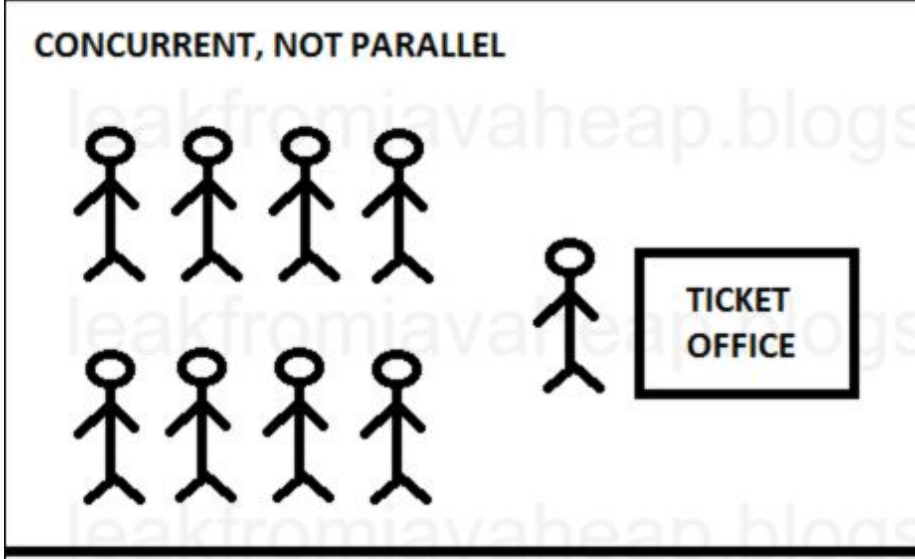
- ❑ Fault tolerance
- ❑ Extensibility
- ❑ Scalability
- ❑ Visibility

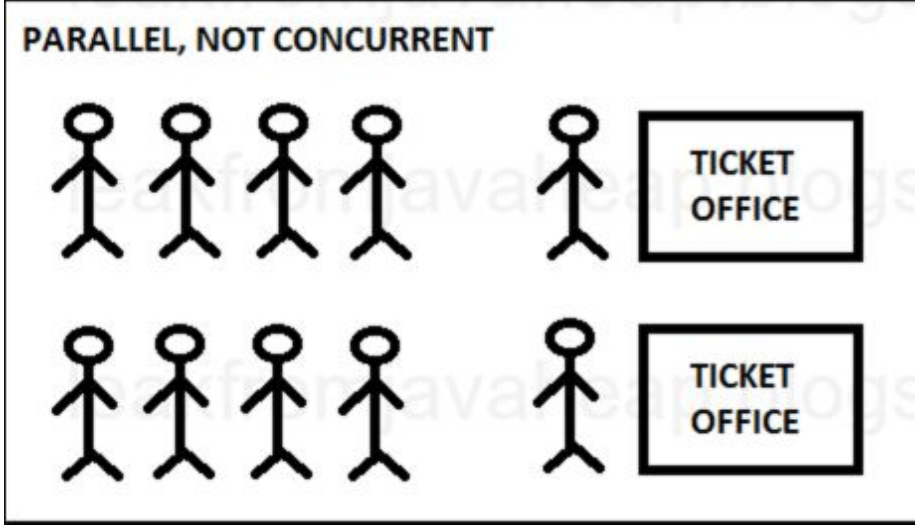
Comparison of available Node.js rate limiters

	rolling rate limiter	node rate limiter	node throttle	node token throttle	**token bucket**
Common / global rate limit	✓	✓	✓	✓	✓
Client or API-specific rate limit	✓	✓	✓	✓	✓
Burst or window rate limit	✓	✓	✓	✓	✓
Quota limit	✗	✗	✗	✗	✓
Concurrent request limit	✗	✗	✗	✗	✗
Persistence of consumed limit	✓	✗	✓	✓	✓
Scalability	✗	✗	✗	✗	✗

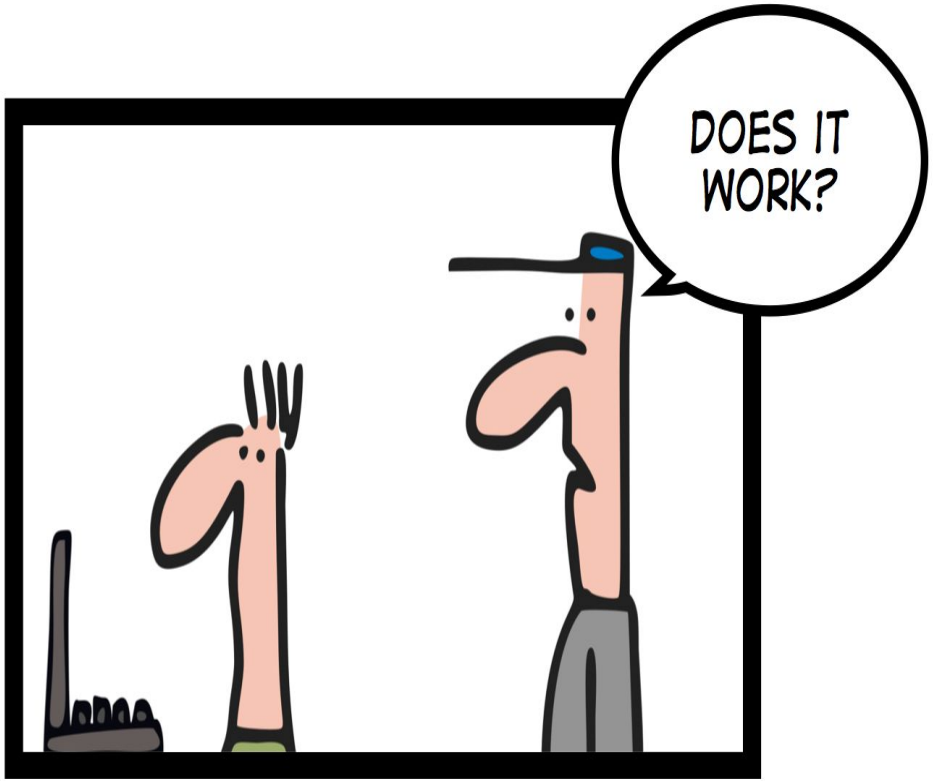
Concurrent Request Limit in Node.js

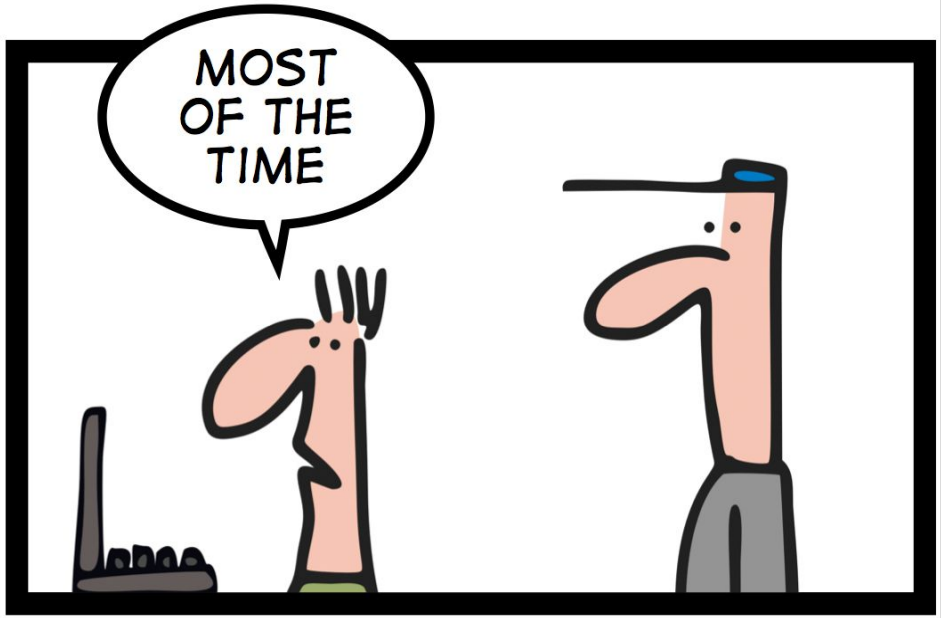
- ❑ Possible? Yes
- ❑ Asynchronous
- ❑ Cluster module for parallelization





SIMPLY EXPLAINED

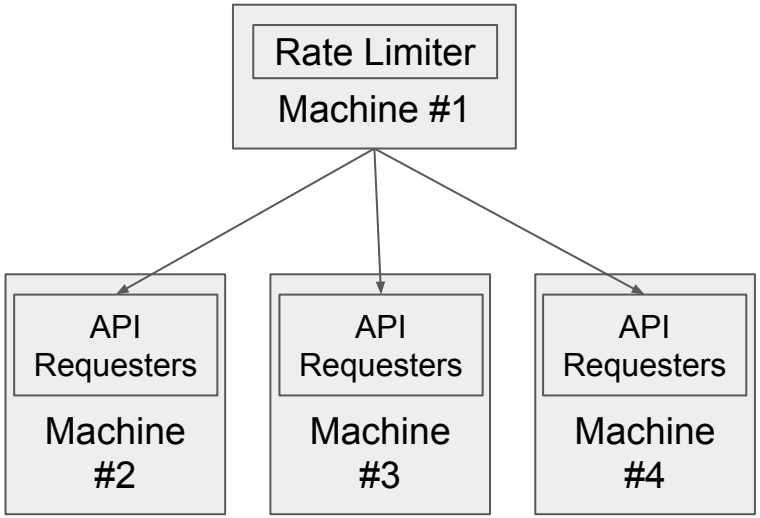


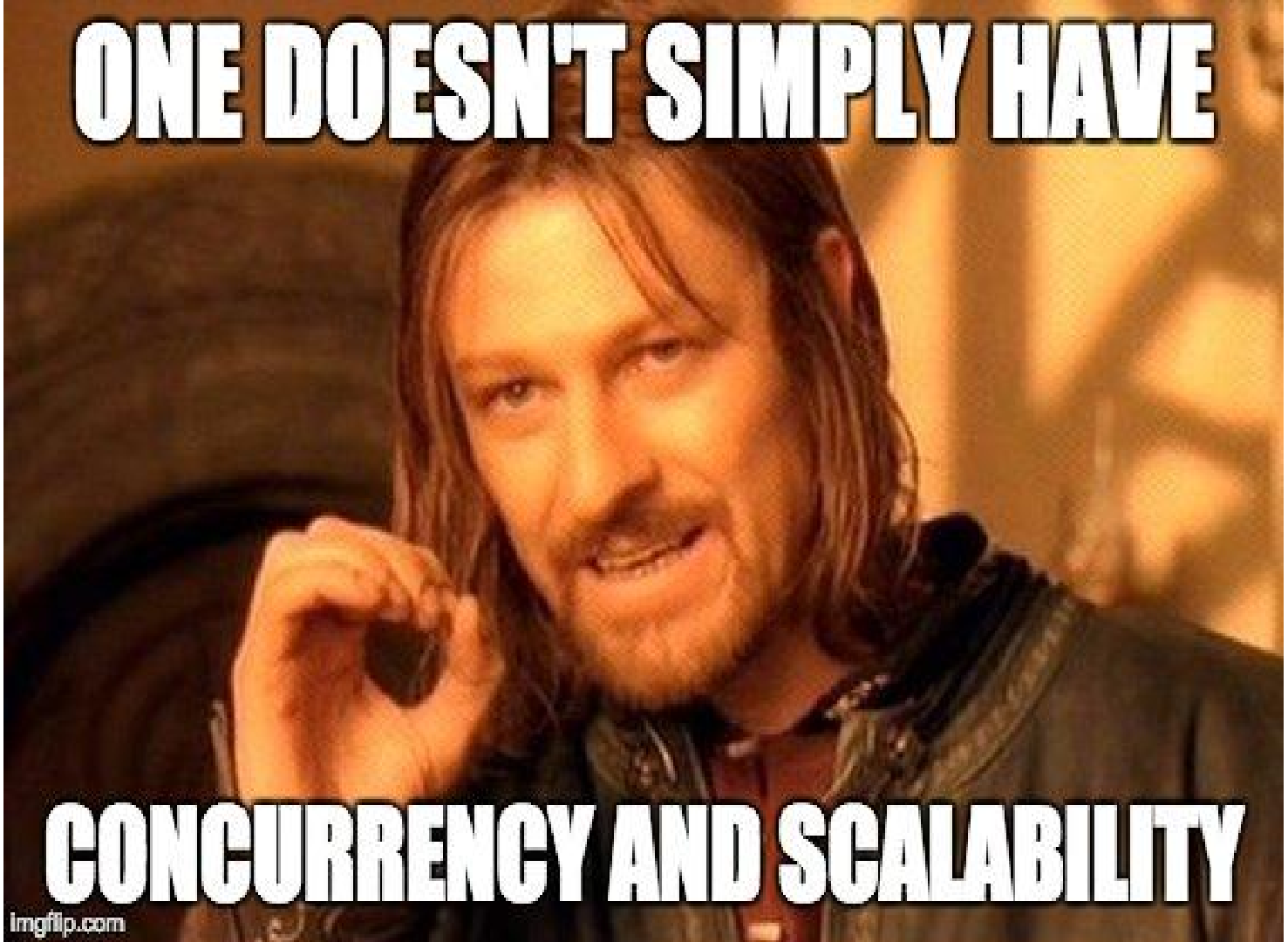


CONCURRENCY

Scalability

- Scenario: Upgraded from 5 QPS to 100 QPS.
- More concurrent requests
- Multiple machines
- How to coordinate?
- More complexity...



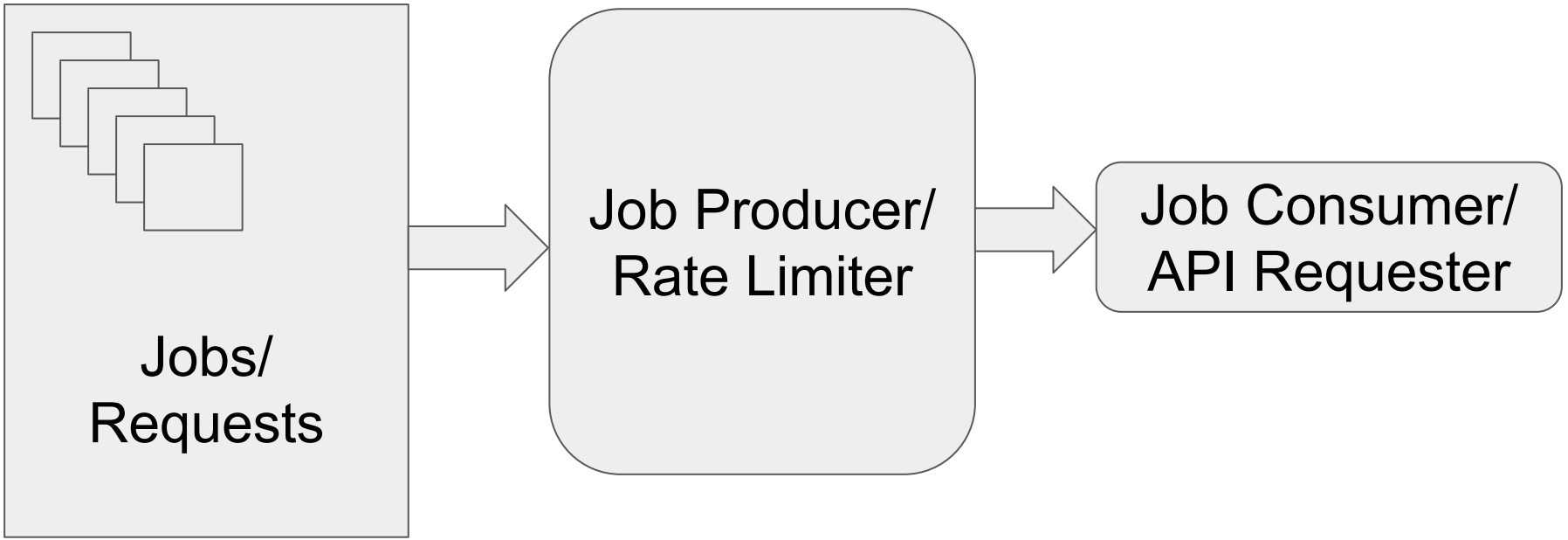

ONE DOESN'T SIMPLY HAVE
CONCURRENCY AND SCALABILITY
imgflip.com

Elixir came to the rescue!

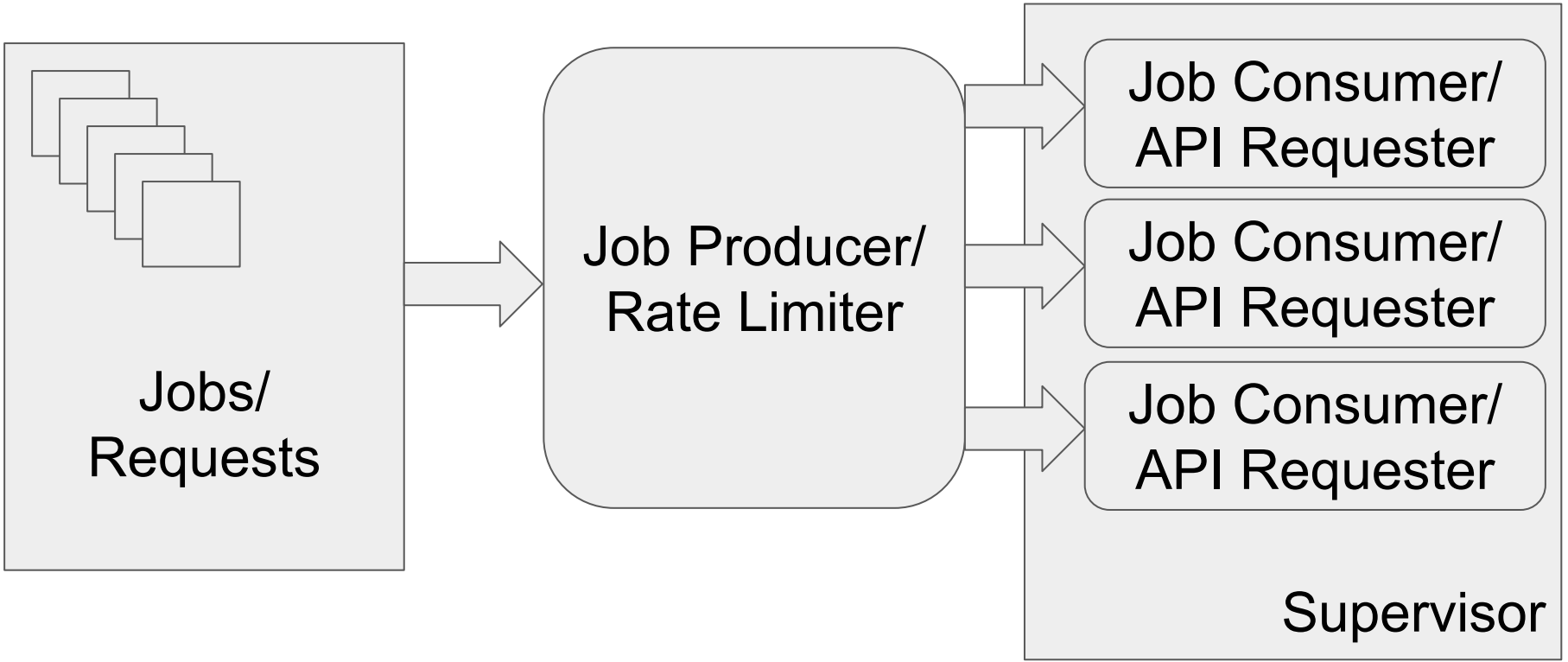
- GenStage: abstracts producer/consumer communication mechanism and back-pressure.
- Rate limiter = Producer
- API Requesters = consumers

```
def add_instance(queue_name, registry_name, job_producer, job_consumer, rate) do
  # implementation goes here.
end
```

Concurrent Request Limit = 1 (Trivial Case)



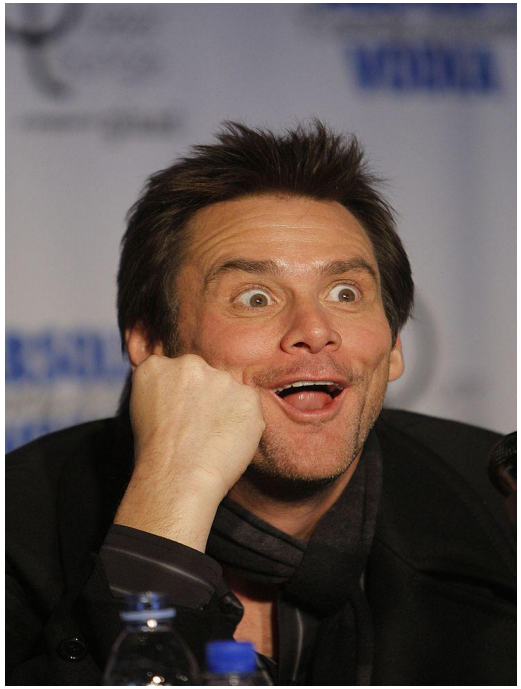
Concurrent request limit = 3

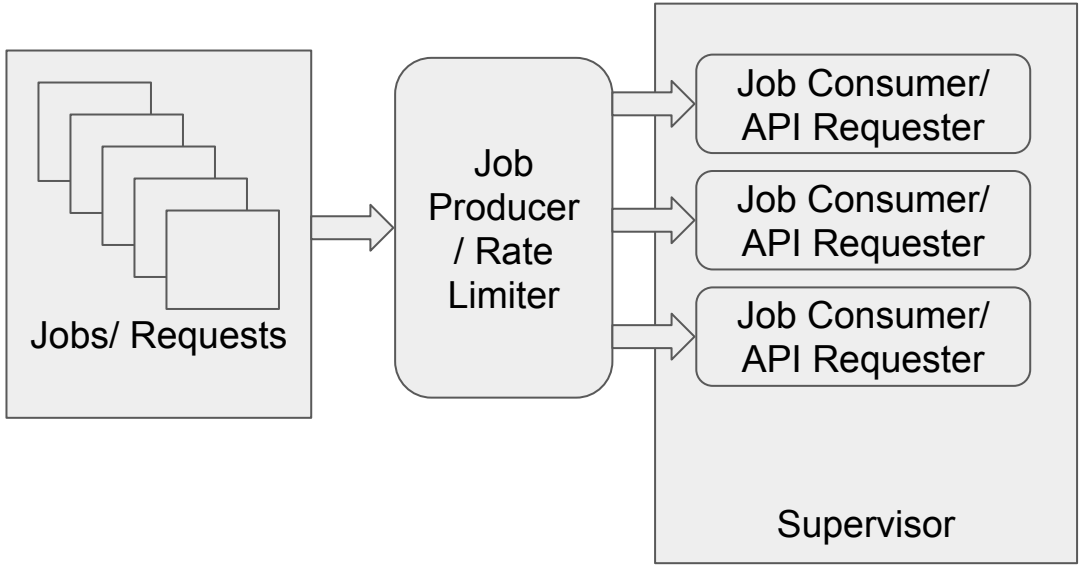


Supervisor Code Example

```elixir
def init({queue_name, job_producer, job_consumer, rate, max_concurrency}) do
  queue_name
  |> get_children(job_producer, job_consumer, rate, max_concurrency)
  |> Supervisor.init(strategy: :one_for_one)
end

defp get_children(queue_name, job_producer, job_consumer, rate, max_concurrency) do
  producer = String.to_atom("#{queue_name}.producer")
  |> (& get_child_spec(producer_module, &1)).()

  consumers = 1..max_concurrency
  |> Enum.to_list
  |> Enum.map(fn n ->
    consumer_name = String.to_atom("#{queue_name}.consumer.#{n}")
    get_child_spec(consumer_module, consumer_name)
  end)

  [producer | consumers]
end
```

Concurrent Request Limit

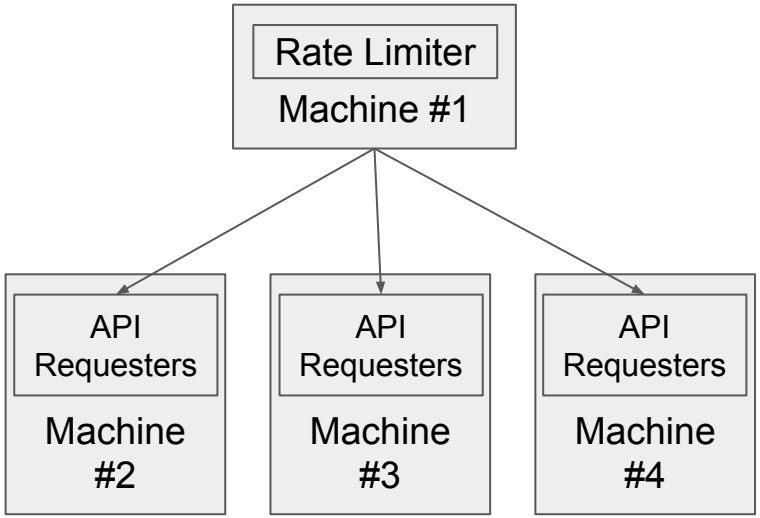




Scalability

- Scenario: Burst rate upgraded from 5 QPS to 100 QPS

- How does Rate Limiter communicate with API Requesters on different machines?
  - Message passing



✓ Concurrent Request Limit

✓ Scalability

Next Challenges

- Plan to open source rate limiter application

- Extend an existing job queue library
  - verk, exq
  - Doesn’t support priority queue and rate limit.

Thank you!

Q&A