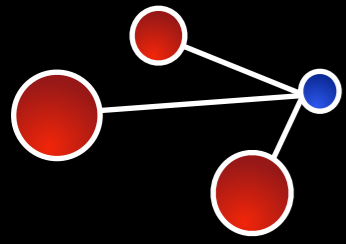


**¡VIVA LA EVOLUTION!**

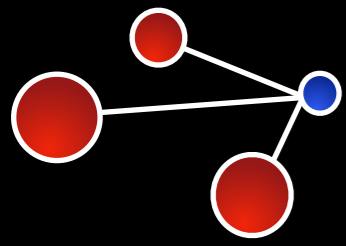
(aka, It's An Evolution!)



# JOHNNY WINN

(@johnny\_rugger)





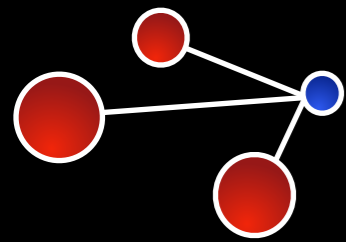
# JOHNNY WINN

(@johnny\_rugger)

(host of the Elixir Fountain)

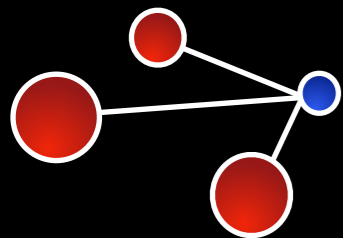
(@elixirfountain)





**! DISCLAIMER!**

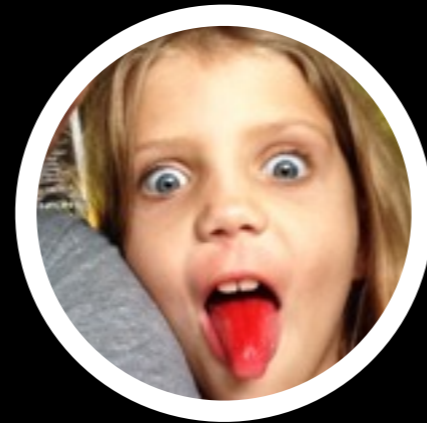
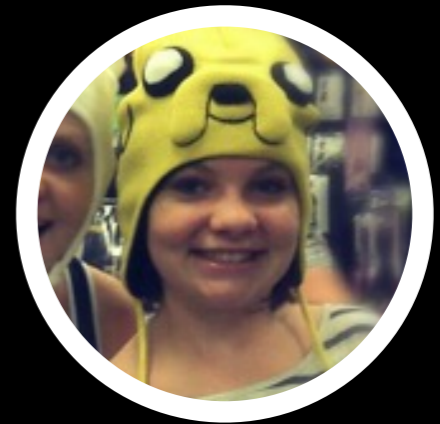
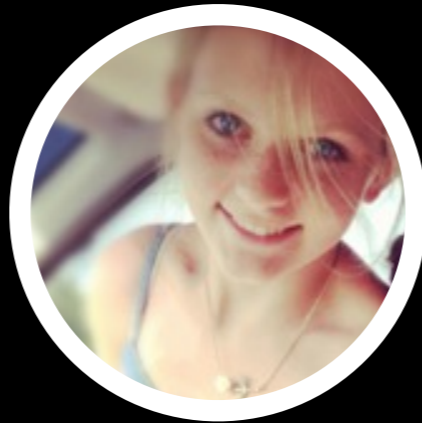
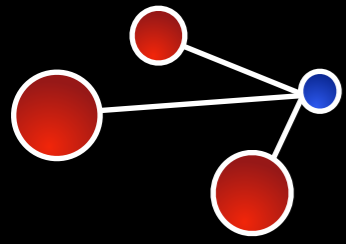
(I am NOT a geneticist)

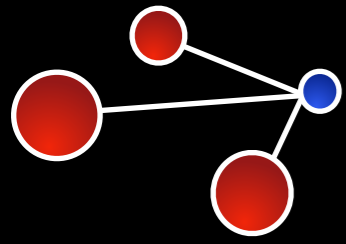


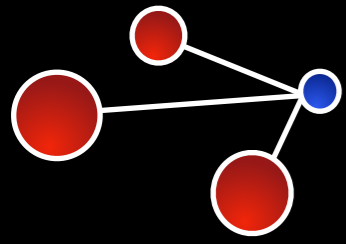
# ! DISCLAIMER!

(I am NOT a geneticist)

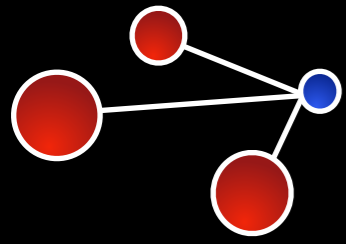
(I don't even play one on TV)

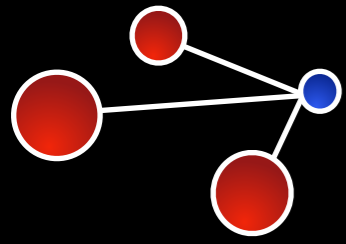




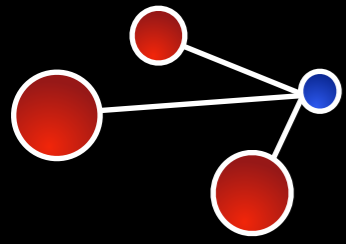




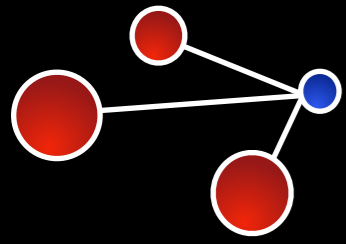




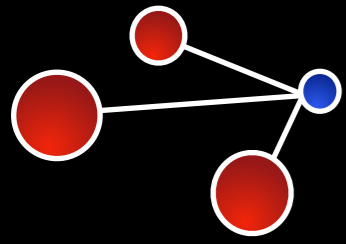
**WHAT?**



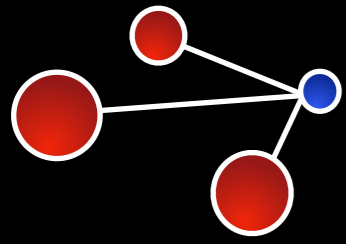
# **(BRIEF) HISTORY OF THE THEORY OF EVOLUTION**



# **(SIMPLE) GENETICS**

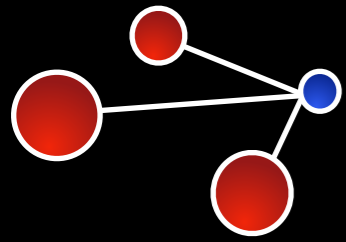


**BUILD (THINGS)**

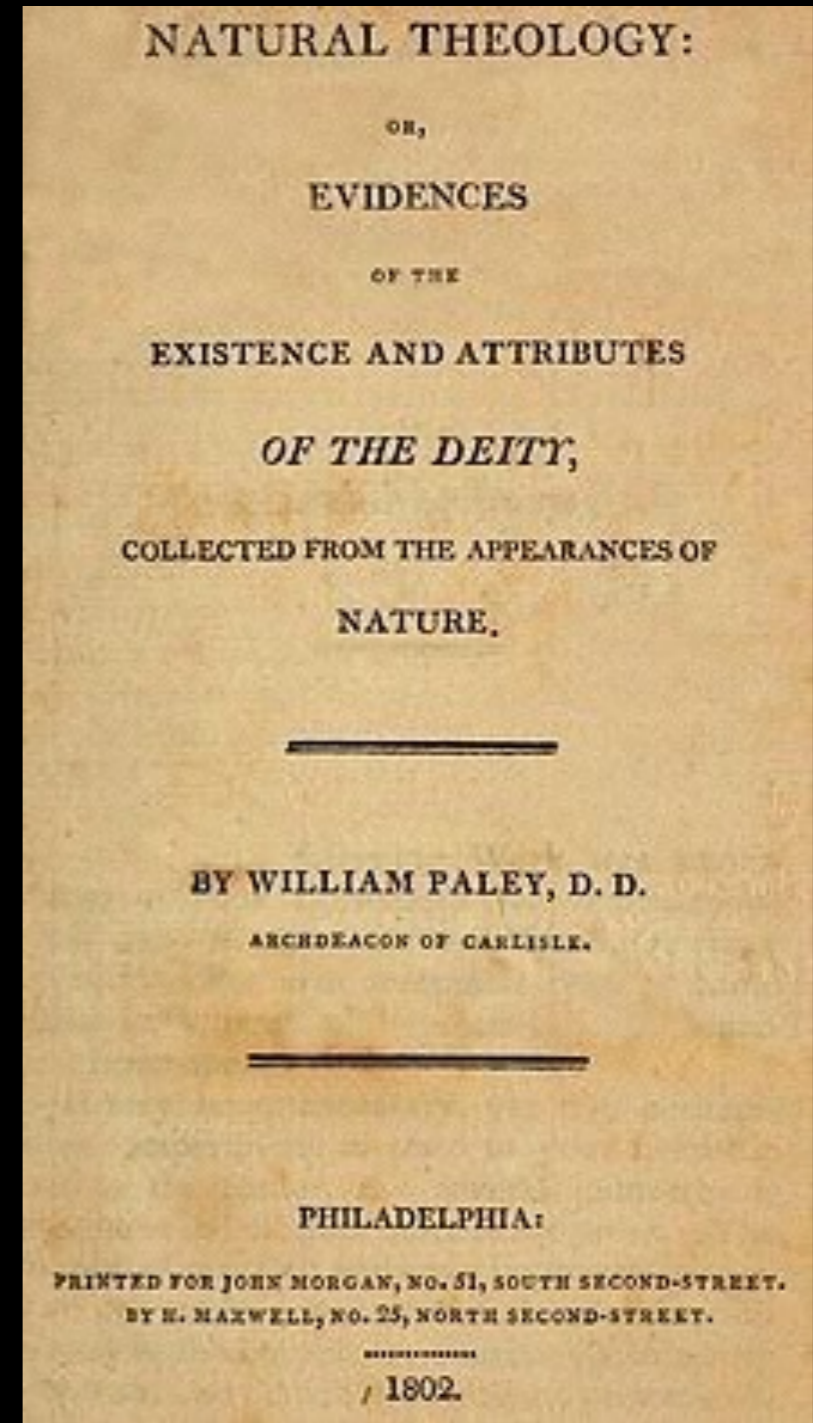


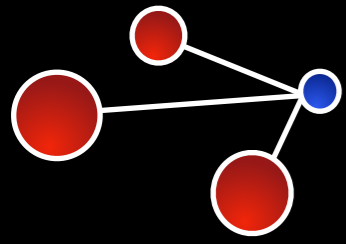
**!EVOLUTION DISCOVERED!**

(in 1831)

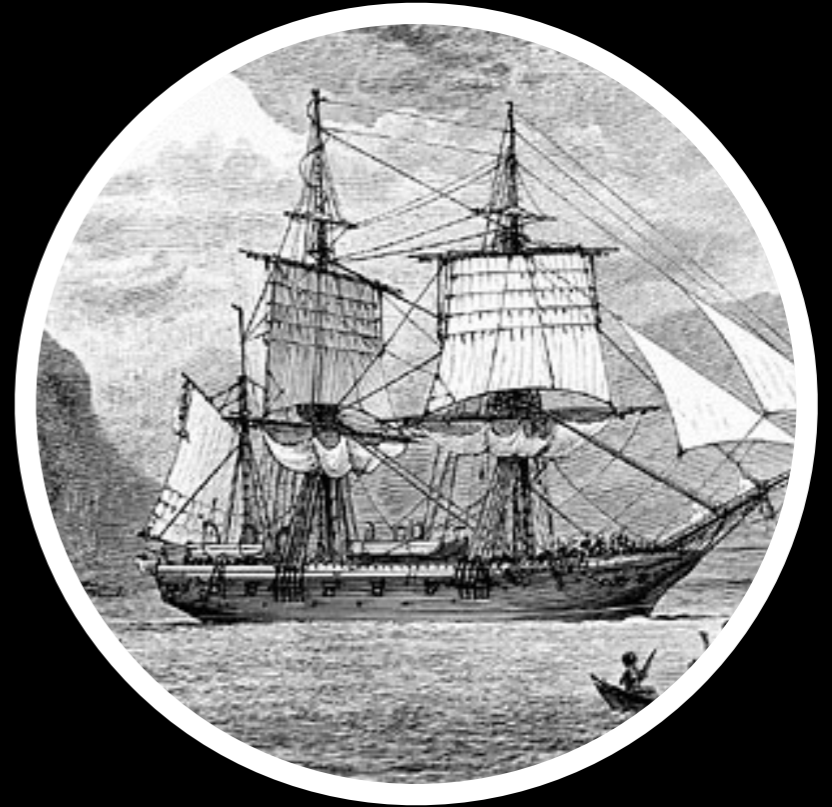


(William Paley)



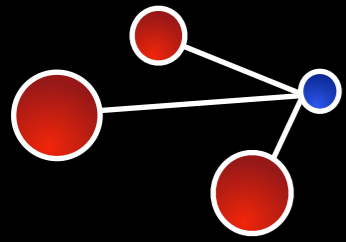


(Charles Darwin)

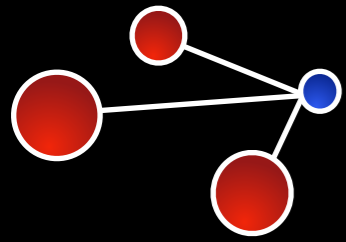


(HMS Beagle)

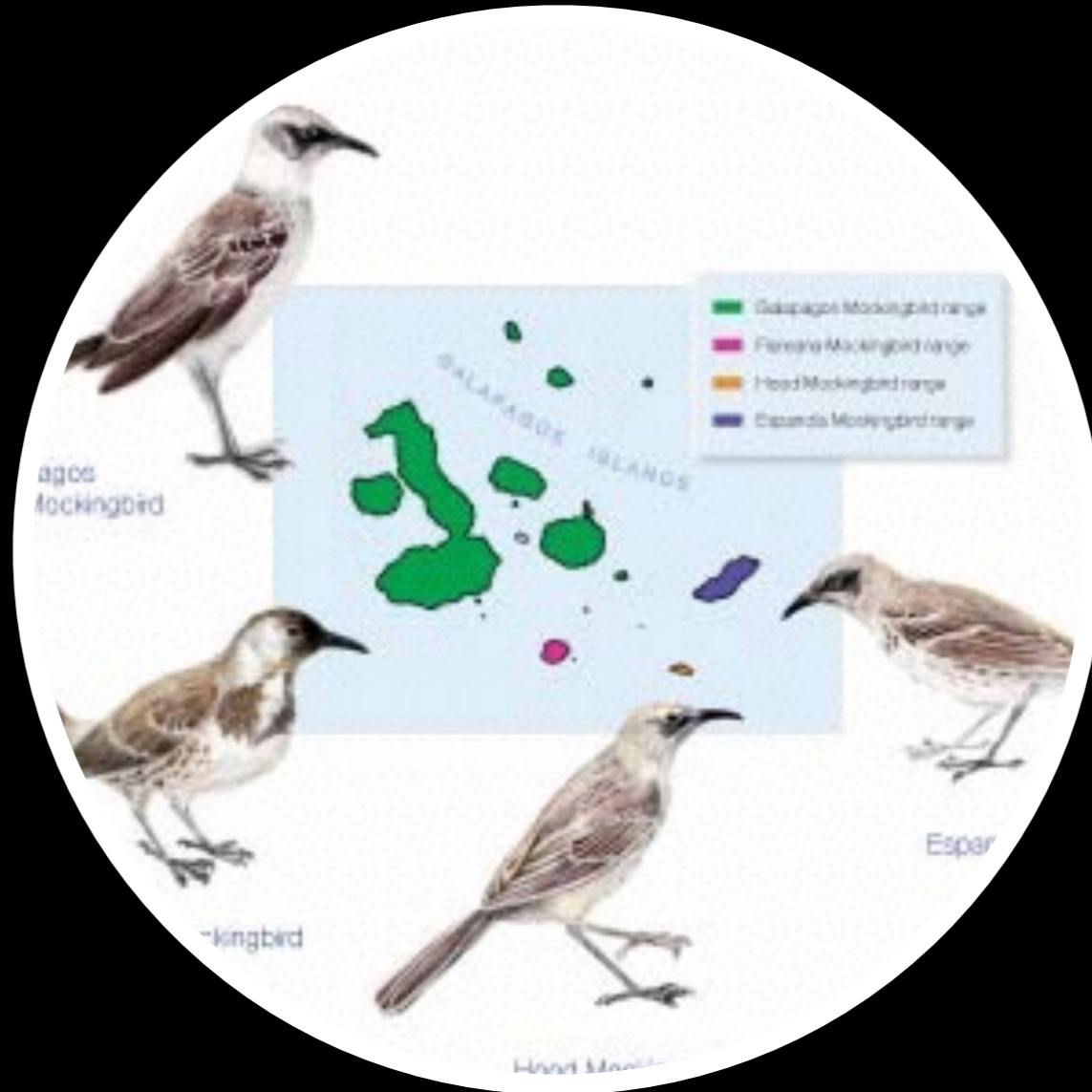
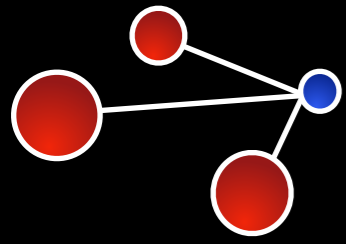


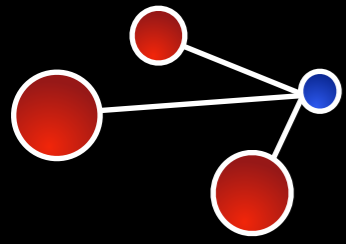


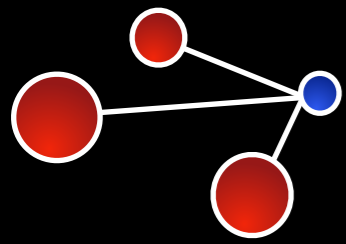
(feature creep?)



(did someone say 3 hour tour?)

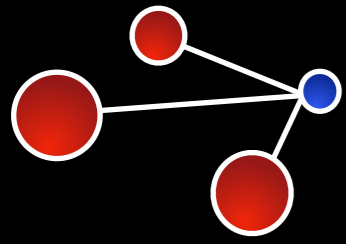




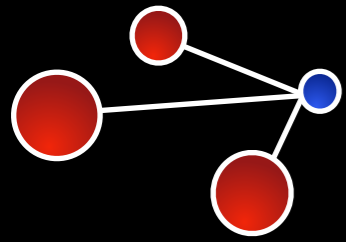


(It is not the strongest of the species that survive,  
nor the most intelligent, but the one most  
responsive to change.)

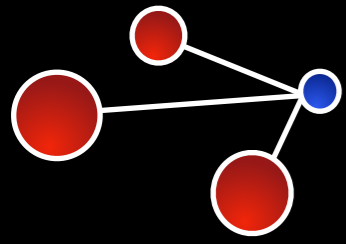
~ **DARWIN**



# **(SIMPLE) GENETICS**

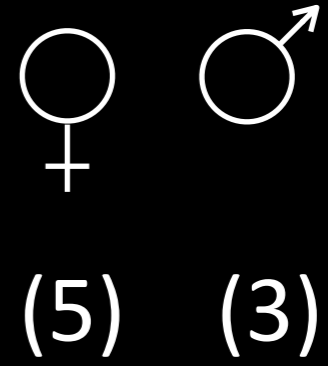
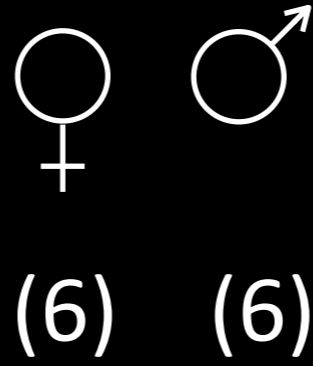
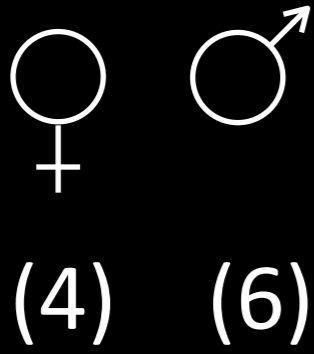
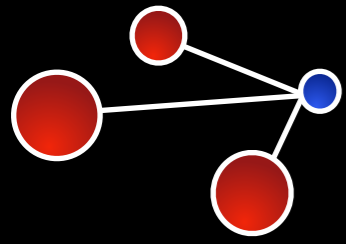


(Gregor Mendel)

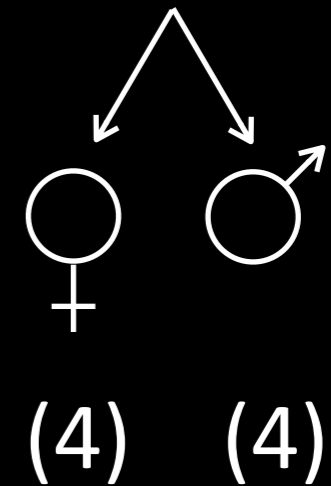
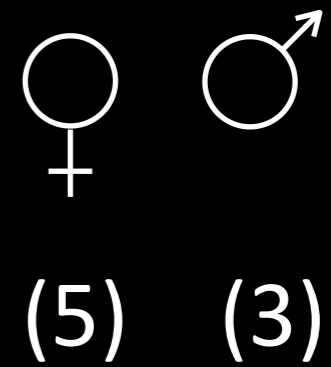
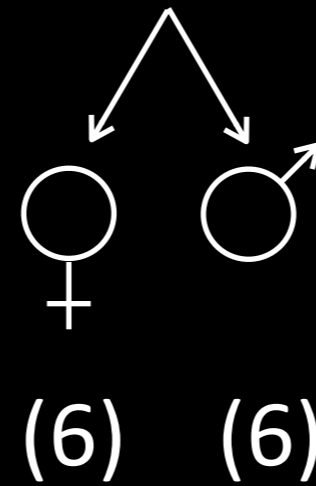
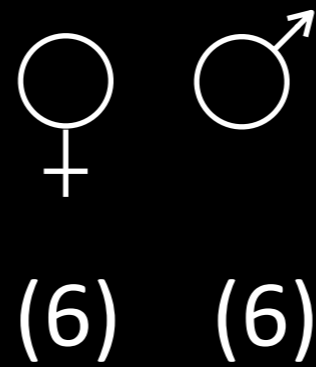
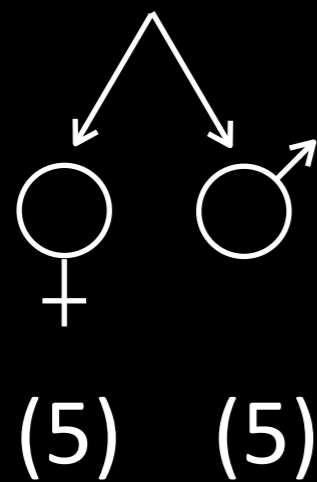
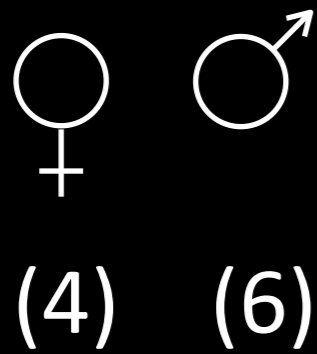
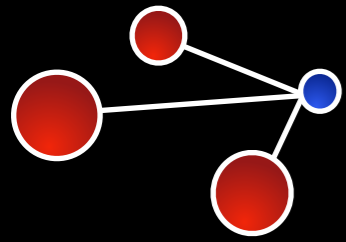


(Why pea plants?)

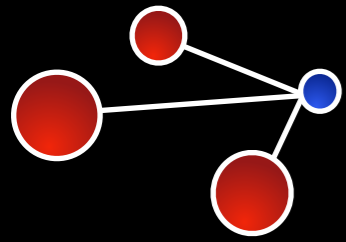

























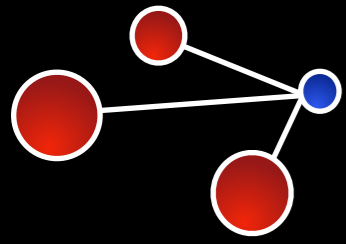
(blended inheritance)



(problem with blended inheritance)

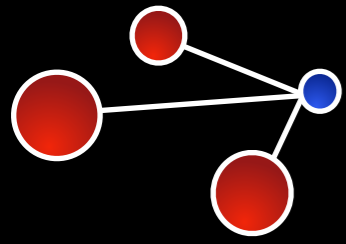


	Flower color	Flower position	Seed color	Seed shape	Pod shape	Pod color	Stem length
P	Purple 	Axial 	Yellow 	Round 	Inflated 	Green 	Tall 
	White 	Terminal 	Green 	Wrinkled 	Constricted 	Yellow 	Dwarf 
F <sub>1</sub>	Purple 	Axial 	Yellow 	Round 	Inflated 	Green 	Tall 



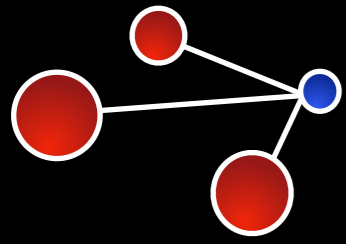
	G	y
G	GG	Gy
y	Gy	yy

(punnet square)



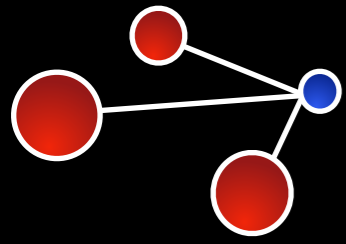
	G	y
G	GG	Gy
y	Gy	yy

(50% Gy, 25% GG, 25% yy)



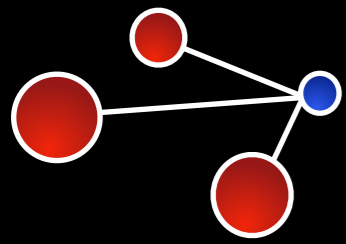
	G	y
G	GG	Gy
G	GG	Gy

(100% Green!)



	G/T	y/t	G/t	y/T
G/T	GG/TT	Gy/Tt	GG/Tt	Gy/TT
y/T	Gy/TT	yy/TT	Gy/Tt	yy/Tt
G/t	GG/Tt	Gy/tt	GG/tt	Gy/Tt
y/t	Gy/Tt	yy/tt	Gy/tt	yy/Tt

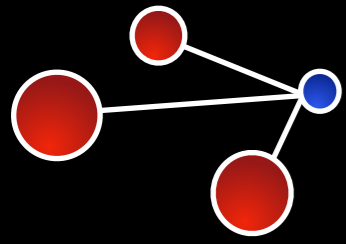
(multiple traits)



	G/T	y/t	G/t	y/T
G/T	GG/TT	Gy/Tt	GG/Tt	Gy/TT
y/T	Gy/TT	yy/TT	Gy/Tt	yy/Tt
G/t	GG/Tt	Gy/tt	GG/tt	Gy/Tt
y/t	Gy/Tt	yy/tt	Gy/tt	yy/Tt

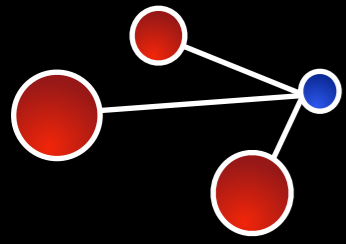
(56% Green/Tall, 19% Yellow/Tall, 19% Yellow/Short, 6% Yellow Short)





# 3 LAWS OF INHERITANCE

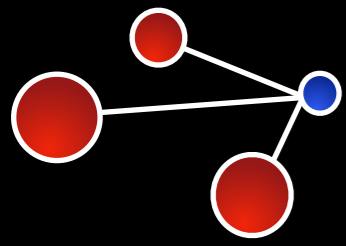
(Law of Segregation)



# 3 LAWS OF INHERITANCE

(Law of Segregation)

(Law of Independent Assortment)

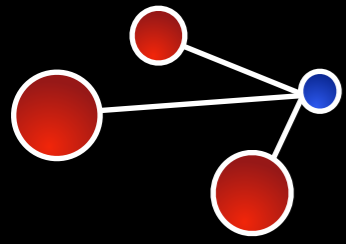


# 3 LAWS OF INHERITANCE

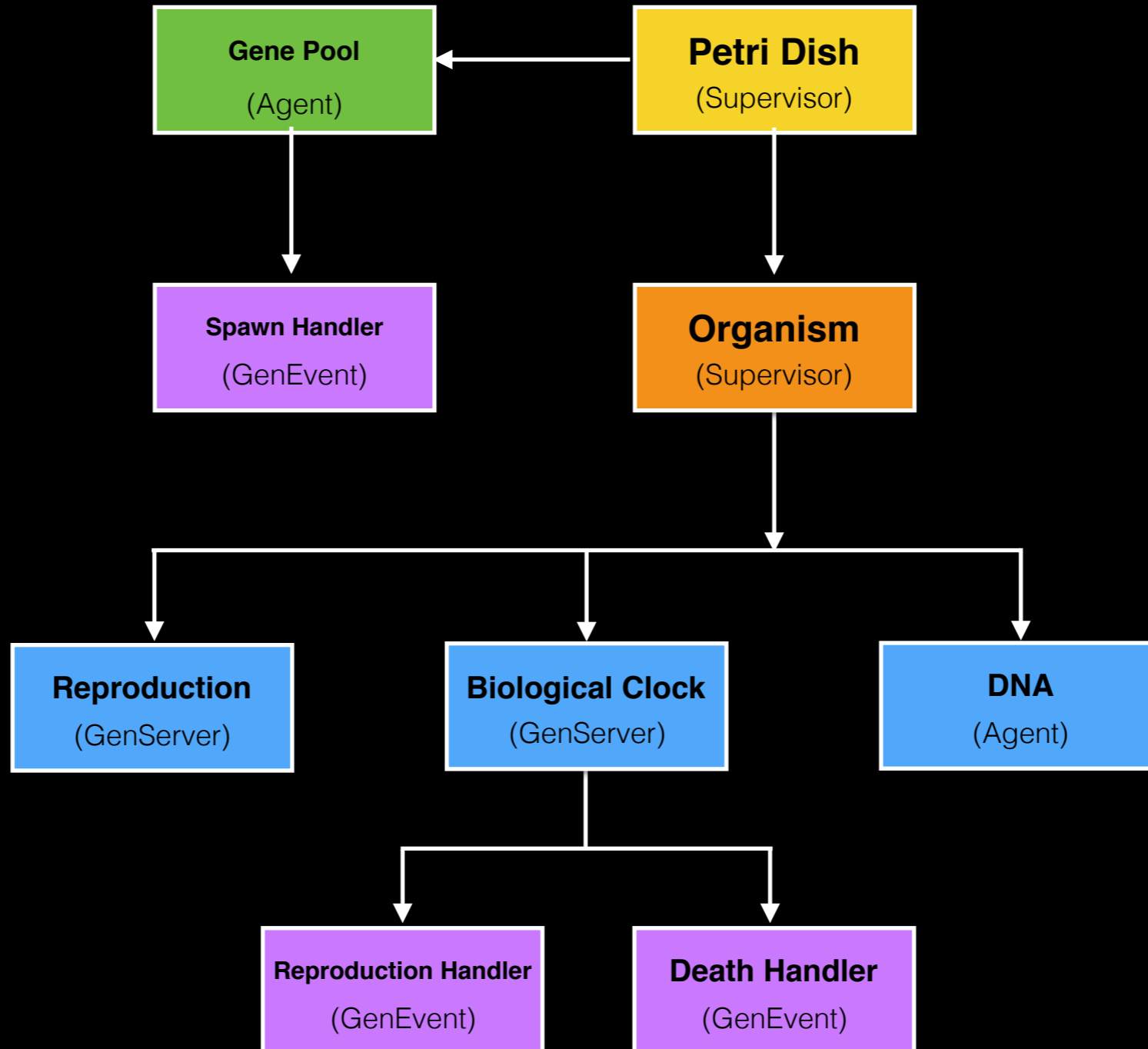
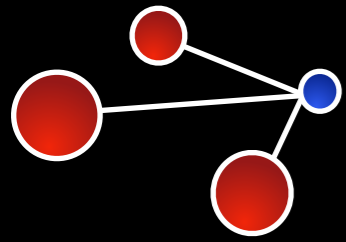
(Law of Segregation)

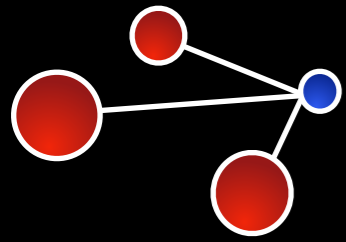
(Law of Independent Assortment)

(Law of Dominance)



**BUILD (THINGS)**





**Petri Dish**  
(Supervisor)



**Organism**  
(Supervisor)



**Biological Clock**  
(GenServer)



**Death Handler**  
(GenEvent)

```
defmodule Darwin.PetriDish do
  use Supervisor

  alias Darwin.Organism
  alias Darwin.DeathHandler

  @moduledoc "The petri dish is where a collection of organisms live"

  def start_link,
    do: Supervisor.start_link(__MODULE__, nil, name: __MODULE__)

  def init(_opts) do
    register_bio_events()

    children = [
      supervisor(Organism, [], restart: :transient)
    ]

    supervise(children, [strategy: :simple_one_for_one])
  end

  def new_organism([x, y]) do
    Supervisor.start_child(__MODULE__, [%{gametes: [x, y]}])
    GenEvent.notify(:darwin, {:new_organism, [x, y]})
  end

  def kill_organism(organism),
    do: Supervisor.terminate_child(__MODULE__, organism.pid)

  defp register_bio_events do
    GenEvent.start_link(name: :bio_events)
    :ok = GenEvent.add_mon_handler(:bio_events, DeathHandler, [])
  end
end
```

```

@process_type "organism"

def start_link(%{gametes: [_x, _y]} = gametes) do
  organism = create_organism(gametes)

  Supervisor.start_link(__MODULE__, organism, name: organism.name)
end

def init(organism) do
  organism = %{organism | pid: self}

  children = [
    worker(Organism.BiologicalClock, [{:organism, organism}]),
  ]

  supervise(children, [strategy: :one_for_one])
end

defp create_organism(%{gametes: gametes}) do
  gametes
  |> define_organism
  |> name_organism
  |> combine_sizes
  |> combine_colors
  |> combine_speeds
end

defp define_organism(gametes),
  do: %Darwin.Organism{gametes: gametes, birth_time: :erlang.system_time()}

defp name_organism(organism),
  do: %{organism | name: Utils.name_process(@process_type, organism)}

defp combine_colors(%{gametes: [%{color: x}, %{color: y}]} = organism),
  do: %{organism | color: [x, y]}

```



```
defmodule Darwin.Organism do
  use Supervisor

  @moduledoc "A single organism"
```

```
defstruct pid: nil,
           name: nil,
           birth_time: nil,
           size: ["S", "s"],
           color: ["C", "c"],
           speed: ["F", "f"],
           gametes: []
```

```
import Darwin.Organism.Utils
```

```
alias Darwin.Organism.Utils
alias Darwin.Organism
```

```
@process_type "organism"
```

```
def start_link(%{gametes: [_x, _y]} = gametes) do
  organism = create_organism(gametes)
```

```
  Supervisor.start_link(__MODULE__, organism, name: organism.name)
end
```

```
def init(organism) do
  organism = %{organism | pid: self}
```

```
  children = [
    worker(Organism.BiologicalClock, [{:organism, organism}]),
  ]
```

```
  supervise(children, [strategy: :one_for_one])
```

```
def init(organism) do
  organism = %{organism | pid: self}

  children = [
    worker(Organism.BiologicalClock, [{:organism, organism}]),
  ]

  supervise(children, [strategy: :one_for_one])
end
```

```
defp create_organism(%{gametes: gametes}) do
  gametes
  |> define_organism
  |> name_organism
  |> combine_sizes
  |> combine_colors
  |> combine_speeds
end
```

```
defp define_organism(gametes),
  do: %Darwin.Organism{gametes: gametes, birth_time: :erlang.system_time()}

defp name_organism(organism),
  do: %{organism | name: Utils.name_process(@process_type, organism)}

defp combine_colors(%{gametes: [%{color: x}, %{color: y}]} = organism),
  do: %{organism | color: [x, y]}

defp combine_speeds(%{gametes: [%{speed: x}, %{speed: y}]} = organism),
  do: %{organism | speed: [x, y]}

defp combine_sizes(%{gametes: [%{size: x}, %{size: y}]} = organism),
  do: %{organism | size: [x, y]}
```

```
@process_type "bio_clock"

def start_link({:organism, organism}),
  do: GenServer.start_link(__MODULE__, organism, name: process_name(organism))

def init(organism) do
  send(self, {:start_biological_clock})

  Process.send_after(self, {:death}, :random.uniform(20000))

  {:ok, organism}
end

def handle_info({:start_biological_clock}, organism),
  do: biological_clock(organism)

def handle_info({:death}, organism),
  do: death(organism)

def handle_info({:reproduce}, organism),
  do: reproduce(organism)

defp death(organism) do
  GenEvent.notify(:bio_events, {:death, organism})
  {:noreply, organism}
end

defp process_name(organism),
  do: Utils.name_process(@process_type, organism)
```

```
defmodule Darwin.DeathHandler do
  use GenEvent

  alias Darwin.PetriDish

  def handle_event({:death, organism}, state) do
    PetriDish.kill_organism(organism)
    {:ok, state}
  end

  def handle_event(_event, state),
    do: {:ok, state}
end
```

## (Petri Dish)

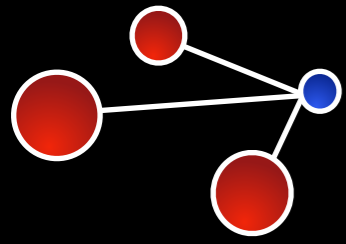


```
def new_organism([x, y]) do
  Supervisor.start_child(__MODULE__, [%{gametes: [x, y]}])
  GenEvent.notify(:darwin, {:new_organism, [x, y]})
end

def kill_organism(organism),
  do: Supervisor.terminate_child(__MODULE__, organism.pid)

defp register_bio_events do
  GenEvent.start_link(name: :bio_events)

  :ok = GenEvent.add_mon_handler(:bio_events, ReproductionHandler, [])
  :ok = GenEvent.add_mon_handler(:bio_events, DeathHandler, [])
  :ok = GenEvent.add_mon_handler(:bio_events, SpawnHandler, [])
end
```



**Petri Dish**  
(Supervisor)



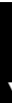
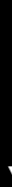
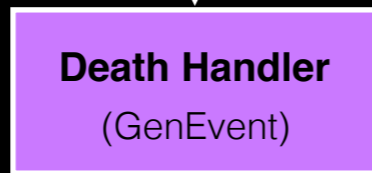
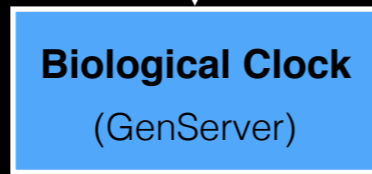
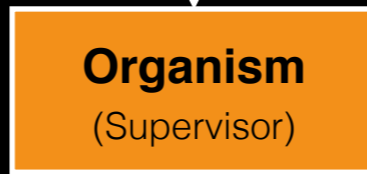
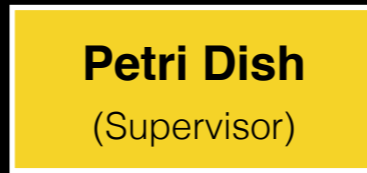
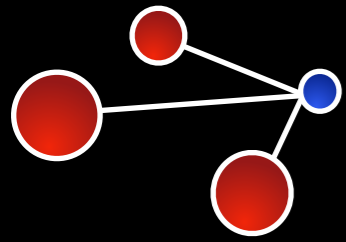
**Organism**  
(Supervisor)



**Biological Clock**  
(GenServer)



**Death Handler**  
(GenEvent)



```
@process_type "dna"

def start_link(%Darwin.Organism{} = organism),
  do: Agent.start_link(fn -> organism end, name: process_name(organism))

def gamete(organism) do
  organism
  |> process_name
  |> Agent.get(&retrieve_gamete/1)
end

def dna(organism) do
  organism
  |> process_name
  |> Agent.get(fn(%{color: color, speed: speed, size: size}) ->
    %{color: join_pairs(color), speed: join_pairs(speed), size: join_pairs(size)}
  end)
end

defp retrieve_gamete(%{color: color, speed: speed, size: size}),
  do: %{color: split_pairs(color), speed: split_pairs(speed), size: split_pairs(size)}

defp split_pairs(phenotypes),
  do: Enum.random(phenotypes)

defp join_pairs(phenotypes),
  do: Enum.join(phenotypes)

defp process_name(organism),
  do: Utils.name_process(@process_type, organism)
```

```
@process_type "dna"

def start_link(%Darwin.Organism{} = organism),
  do: Agent.start_link(fn -> organism end, name: process_name(organism))

def gamete(organism) do
  organism
  |> process_name
  |> Agent.get(&retrieve_gamete/1)
end

def dna(organism) do
  organism
  |> process_name
  |> Agent.get(fn(%{color: color, speed: speed, size: size}) ->
    %{color: join_pairs(color), speed: join_pairs(speed), size: join_pairs(size)}
  end)
end
```

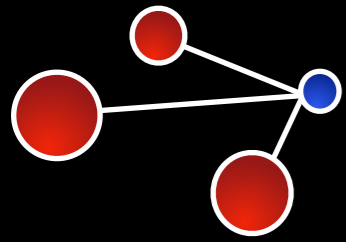
```
defp retrieve_gamete(%{color: color, speed: speed, size: size}),
  do: %{color: split_pairs(color), speed: split_pairs(speed), size: split_pairs(size)}
```

```
defp split_pairs(phenotypes),
  do: Enum.random(phenotypes)
```

```
defp join_pairs(phenotypes),
  do: Enum.join(phenotypes)
```

```
defp process_name(organism),
  do: Utils.name_process(@process_type, organism)
```





**Petri Dish**  
(Supervisor)

**Organism**  
(Supervisor)

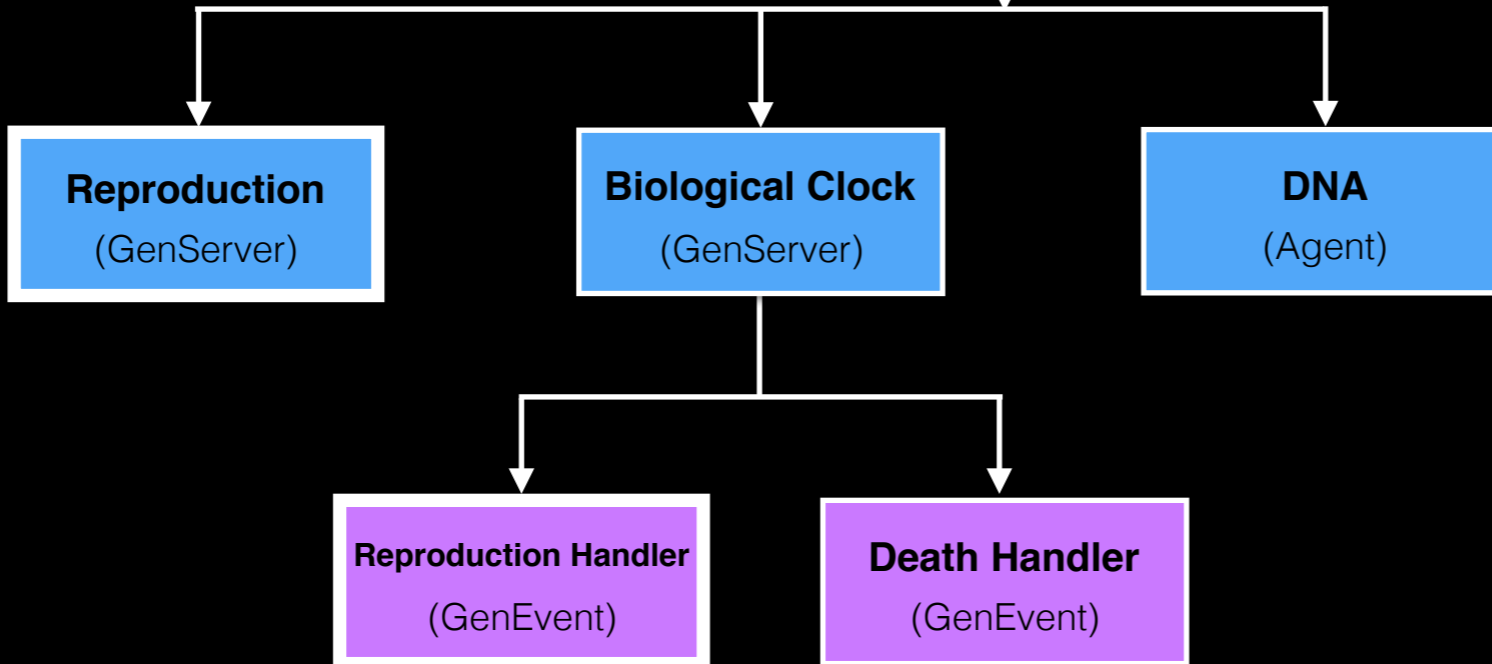
**Reproduction**  
(GenServer)

**Biological Clock**  
(GenServer)

**DNA**  
(Agent)

**Reproduction Handler**  
(GenEvent)

**Death Handler**  
(GenEvent)



```
def handle_info({:start_biological_clock}, organism),
  do: biological_clock(organism)

def handle_info({:death}, organism),
  do: death(organism)

def handle_info({:reproduce}, organism),
  do: reproduce(organism)

defp biological_clock(%{speed: ["F", "F"]} = organism) do
  Process.send_after(self, {:reproduce}, :random.uniform(2500))
  {:noreply, organism}
end

defp biological_clock(%{speed: ["f", "f"]} = organism) do
  Process.send_after(self, {:reproduce}, :random.uniform(7500))
  {:noreply, organism}
end

defp biological_clock(organism) do
  Process.send_after(self, {:reproduce}, :random.uniform(5000))
  {:noreply, organism}
end

defp death(organism) do
  GenEvent.notify(:bio_events, {:death, organism})
  {:noreply, organism}
end

defp reproduce(organism) do
  GenEvent.notify(:bio_events, {:reproduce, organism})
  biological_clock(organism)
end
```

```

defmodule Darwin.ReproductionHandler do
  use GenEvent

  alias Darwin.GenePool
  alias Darwin.Organism.DNA

  def handle_event({:reproduce, organism}, state) do
    GenePool.join_pool({organism.pid, DNA.gamete(organism)})
    {:ok, state}
  end

  def handle_event(_event, state),
    do: {:ok, state}
end

```

## (Petri Dish)

```

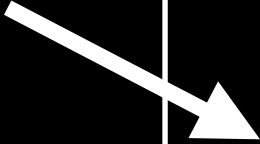
def new_organism([x, y]) do
  Supervisor.start_child(__MODULE__, [%{gametes: [x, y]}])
  GenEvent.notify(:darwin, {:new_organism, [x, y]})
end

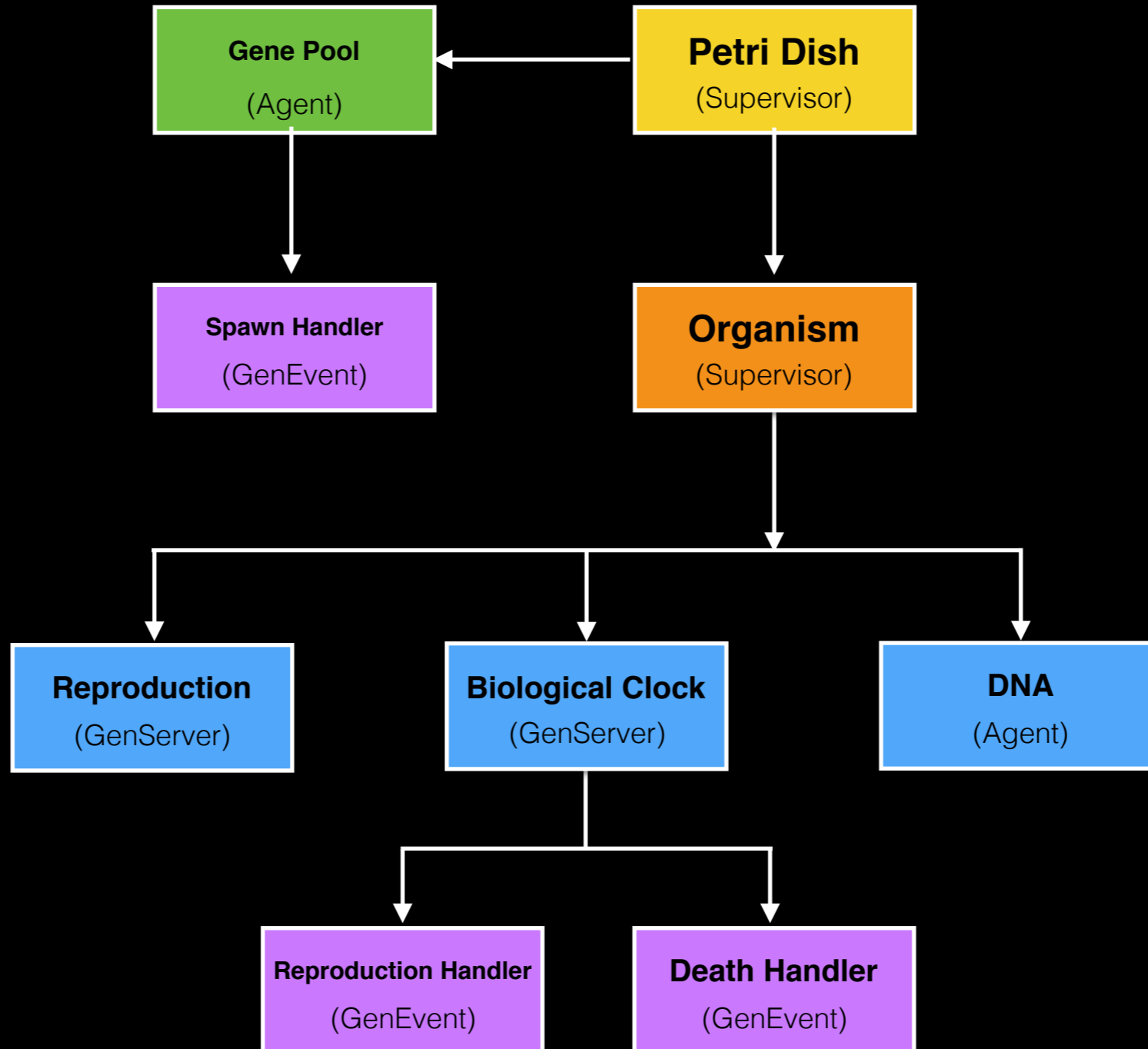
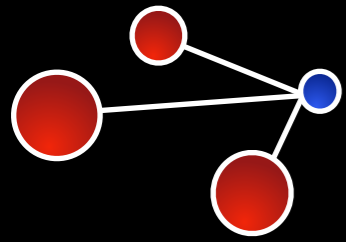
def kill_organism(organism),
  do: Supervisor.terminate_child(__MODULE__, organism.pid)

defp register_bio_events do
  GenEvent.start_link(name: :bio_events)

  :ok = GenEvent.add_mon_handler(:bio_events, ReproductionHandler, [])
  :ok = GenEvent.add_mon_handler(:bio_events, DeathHandler, [])
end

```





```
defstruct parent: nil,  
          gamete: nil  
  
@fitness %{color: "C", speed: "F", size: "s"}  
  
def start_link,  
  do: Agent.start_link(fn() -> [] end, name: __MODULE__)  
  
def current_pool,  
  do: Agent.get(__MODULE__, fn(pool) -> pool end)  
  
def join_pool({parent, gamete}) do  
  check_for_mate  
  |> take_action_on_match({parent, gamete})  
end  
  
defp check_for_mate,  
  do: Agent.get(__MODULE__, &best_match_or_random/1)  
  
defp best_match_or_random([]),  
  do: []  
  
defp best_match_or_random(pool),  
  do: best_match_or_random(pool, [])  
  
defp best_match_or_random([%Darwin.GenePool{gamete: @fitness} = match | _rest], _acc),  
  do: match  
  
defp best_match_or_random([gamete | rest], acc),  
  do: best_match_or_random(rest, [gamete | acc])  
  
defp best_match_or_random([], acc),  
  do: Enum.random(acc)
```

```

defp check_for_mate,
  do: Agent.get(__MODULE__, &best_match_or_random/1)

defp best_match_or_random([]),
  do: []

defp best_match_or_random(pool),
  do: best_match_or_random(pool, [])

defp best_match_or_random([%Darwin.GenePool{gamete: @fitness} = match | _rest], _acc),
  do: match

defp best_match_or_random([gamete | rest], acc),
  do: best_match_or_random(rest, [gamete | acc])

defp best_match_or_random([], acc),
  do: Enum.random(acc)

defp take_action_on_match([], {parent, gamete}) do
  Agent.update(__MODULE__, fn(pool) ->
    [%Darwin.GenePool{parent: parent, gamete: gamete} | pool]
  end)
end

defp take_action_on_match(%Darwin.GenePool{gamete: y} = mate, {parent, x}) do
  GenEvent.notify(:bio_events, {:spawn_organism, [x, y]})

  Agent.update(__MODULE__, fn(pool) ->
    Enum.filter(pool, &(&1 != mate))
  end)
end

```

```
defmodule Darwin.SpawnHandler do
  use GenEvent

  alias Darwin.PetriDish

  def handle_event({:spawn_organism, [_x, _y] = gametes}, state) do
    PetriDish.new_organism(gametes)
    {:ok, state}
  end

  def handle_event(_event, state),
    do: {:ok, state}
end
```

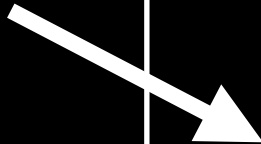
## (Petri Dish)

```
def new_organism([x, y]) do
  Supervisor.start_child(__MODULE__, [%{gametes: [x, y]}])
  GenEvent.notify(:darwin, {:new_organism, [x, y]})
end

def kill_organism(organism),
  do: Supervisor.terminate_child(__MODULE__, organism.pid)

defp register_bio_events do
  GenEvent.start_link(name: :bio_events)

  :ok = GenEvent.add_mon_handler(:bio_events, ReproductionHandler, [])
  :ok = GenEvent.add_mon_handler(:bio_events, DeathHandler, [])
  :ok = GenEvent.add_mon_handler(:bio_events, SpawnHandler, [])
end
```



(but does it work?)





```
# Colors Green (G), Yellow (y)  
# Size Tall (T), short(t)
```

```
x1 = %{color: "G", speed: "f", size: "t"}  
y1 = %{color: "y", speed: "F", size: "T"}
```

```
Darwin.PetriDish.new_organism [x1, y1]
```

(let's see it)



```
# Colors Green (G), Yellow (y)  
# Size Tall (T), short(t)
```

```
x1 = %{color: "G", speed: "f", size: "t"}  
y1 = %{color: "y", speed: "F", size: "T"}
```

```
x2 = %{color: "y", speed: "F", size: "t"}  
y2 = %{color: "G", speed: "f", size: "T"}
```

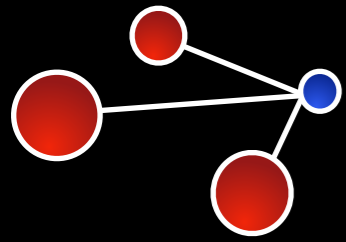
```
x3 = %{color: "G", speed: "f", size: "t"}  
y3 = %{color: "G", speed: "F", size: "T"}
```

```
Darwin.PetriDish.new_organism [x1, y1]
```

```
Darwin.PetriDish.new_organism [x2, y2]
```

```
Darwin.PetriDish.new_organism [x3, y3]
```

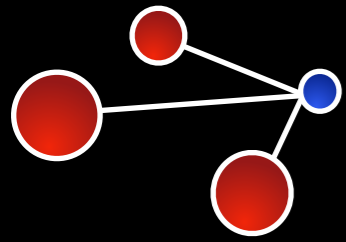
(more organisms!)



(Gregor Mendel)



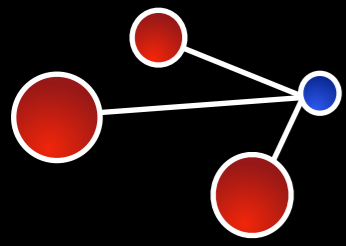
(Charles Darwin)



(Gregor Mendel)



(Charles Darwin)



# #MYELIXIRSTATUS

(it's about sharing our discoveries)