Background Jobs from Scratch

Hans Schnedlitz

Hans Schnedlitz

Freelance Ruby/RoR Engineer 💎

- Vienna.rb Organizer 🇦🇹
- RubyConf Austria Co-Organizer 🎉
- @hansschnedlitz 🦋
- hansschnedlitz.com 🌐


RUBY CONF AT
29-31.05. 2026

Dave Thomas

Author of The Pragmatic Programmer, Programming Ruby, Agile Web Development With Rails, Programming Elixir...

Chad Fowler

CTO & General Partner, BlueYard

José Valim

Creator of the Elixir programming language and Chief Adoption Officer at Dashbit, the company behind Tidewave and Livebook.

Zuzanna Kusznir (MC)

Software engineer at Momentum | wroclove.rb host

Iman Jahić Hošić

Piano/Klavier

Nadir Hošić

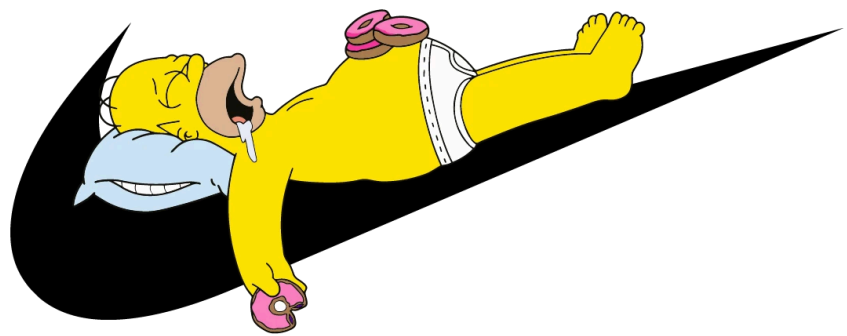
Piano/Klavier

Julia Siedl

Piano/Klavier

Background Jobs from Scratch

JUST DO IT



LATER.

Background Jobs

... there's a lot of them 💎

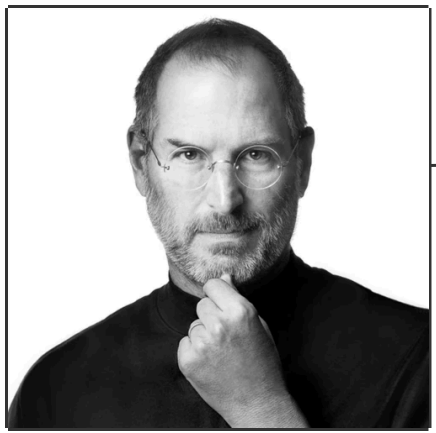
- Sidekiq
- Resque
- Delayed Job
- Solid Queue
- GoodJob
- Que
- Shoryken
- Sucker Punch

Builing SimpleJob

1. A Quick Overview
2. Implementing SimpleJob
3. ~~Profit~~ 💸
4. Why you *shouldn't* implement background jobs from scratch 😈

Your Jobs, Workers, and the In-Between

Jobs



??

Workers

Let’s build SimpleJob

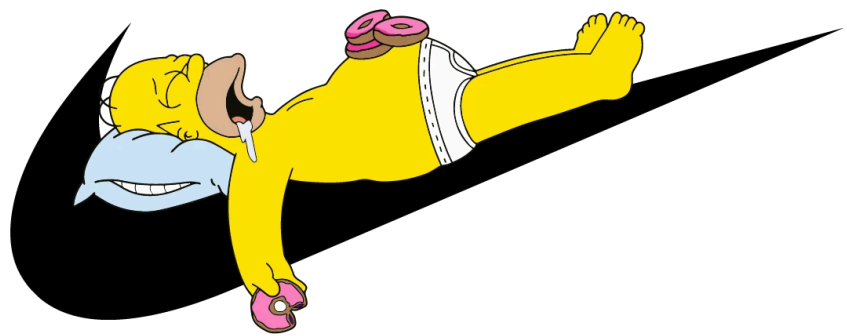
Jobs & Queue Adapters

```ruby
class MyJob < ApplicationJob
  def perform
    puts "I work hard and party harder! 💪"
  end
end

MyJob.perform_later
```

```ruby
# lib/simple_job/adapter.rb
module ActiveJob
  module QueueAdapters
    class SimpleJobAdapter
      def enqueue(active_job)
        # Enqueue the job... but how? 🤔
      end
    end
  end
end
```

```ruby
# config/initializers/simple_job.rb
Rails.application.config.active_job.queue_adapter = :simple_job
```

JUST DO IT



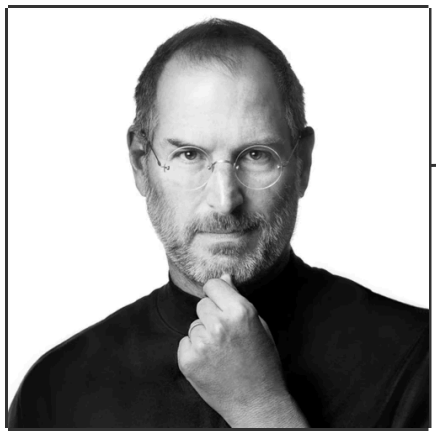
LATER.

Workers

```ruby
module SimpleJob
  class Worker
    def start
      puts "SimpleJob Worker starting..."
      run
    end

    def run
      loop do
        break if @shutdown
        jobs = poll
        sleep(0.1) if jobs == 0
      end

      puts "\nSimpleJob Worker shutting down..."
    end

    def poll
      # Get some jobs and work them!
    end
  end
end
```

The In-Between

Jobs



??

Workers

Let’s Talk Tables

Write to enqueue, query to dequeue...

```ruby
class CreateSimpleJobTables < ActiveRecord::Migration[7.2]
  def change
    create_table :simple_jobs do |t|
      t.string :class_name, null: false
      t.text :arguments
      t.datetime :finished_at
      t.timestamps
    end
  end
end
```

ActiveRecord 🤝 SimpleJob

```ruby
module SimpleJob
  class Job < ApplicationRecord
    self.table_name = "simple_jobs"

    serialize :arguments, coder: JSON

    def self.enqueue(class_name, args)
      create!(
        class_name: class_name,
        arguments: args,
      )
    end
  end
end
```

Queue Adapters

```
module ActiveJob
  module QueueAdapters
    class SimpleJobAdapter
      def enqueue(active_job)
        end
      end
    end
  end
```

```
module ActiveJob
  module QueueAdapters
    class SimpleJobAdapter
      def enqueue(active_job)
+       SimpleJob::Job.enqueue(
+         active_job.class.name,
+         active_job.serialize["arguments"],
+       )
      end
    end
  end
end
```

Workers

```
module SimpleJob
  class Worker
    def poll
      end
  end
end
```

```
module SimpleJob
  class Worker
    def poll
+     job = SimpleJob::Job.where(finished_at: nil).first
+     return 0 unless job
+
+     ActiveRecord::Base.transaction do
+       execute(job)
+       job.update!(finished_at: Time.current)
+     end
+     1
+   end
+
+   def execute(job)
+     job_class = job.class_name.constantize
+     job_instance = job_class.new
+     job_instance.perform(*arguments)
    end
  end
end
```

This Sucks 🤮

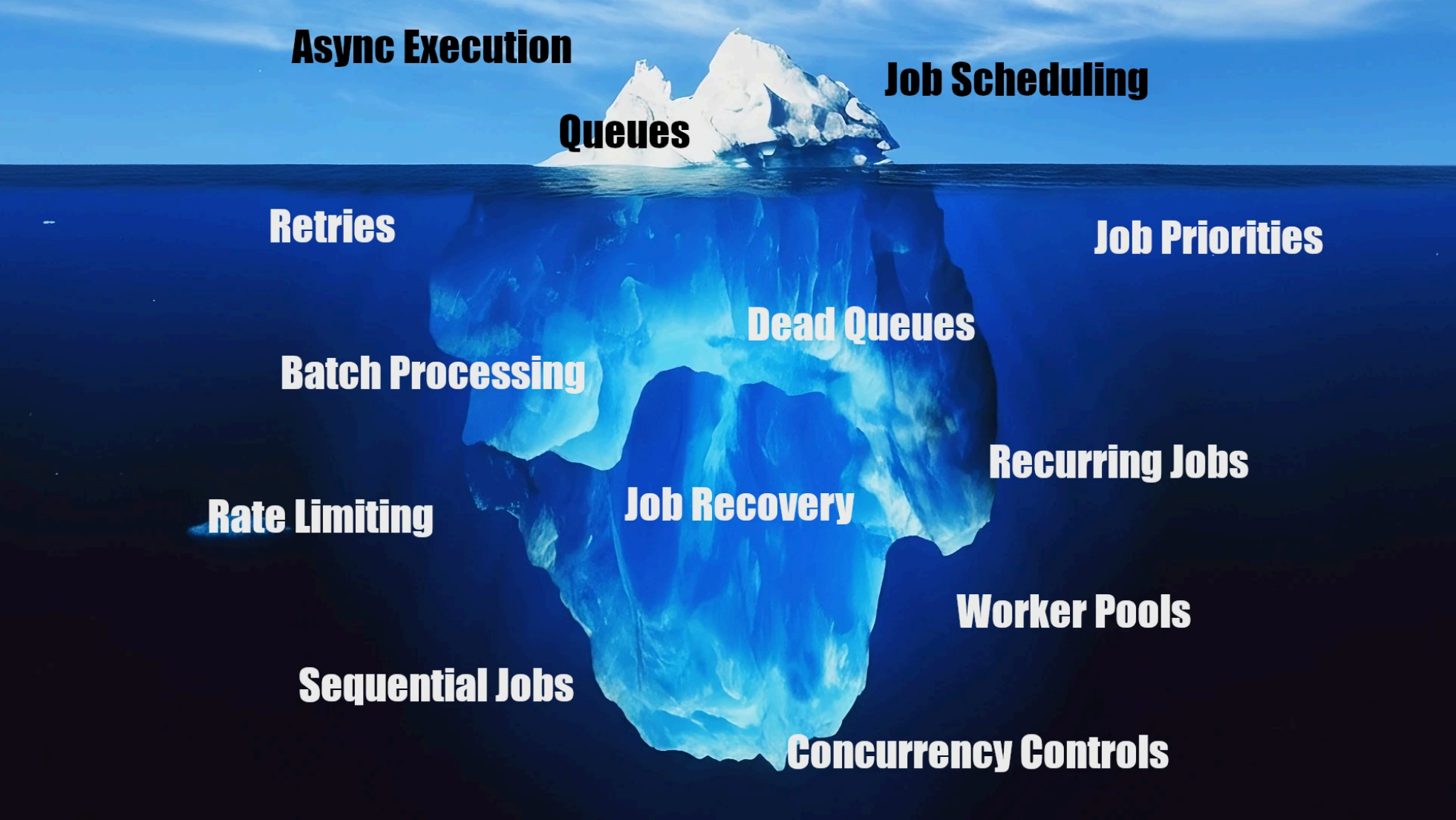
What about my *Features*?

❌ Scheduling
❌ Queues
❌ Job priority
❌ Recurring jobs
❌ Retries

This Sucks 🤮

What about *everything else*?

❌ Multithreading
❌ Job recovery
❌ Process heartbeats and supervisors
❌ Query performance?!?


Async Execution
Job Scheduling
Queues
Retries
Job Priorities
Dead Queues
Batch Processing
Recurring Jobs
Job Recovery
Rate Limiting
Worker Pools
Sequential Jobs
Concurrency Controls

The Problem with Polling

This is what treating your database as a queue gets you 🤦

```ruby
class CreateSimpleJobTables < ActiveRecord::Migration[7.2]
  def change
    create_table :simple_jobs do |t|
      t.string :class_name, null: false
      t.text :arguments
      t.datetime :finished_at
      t.timestamps
    end
  end
end
```

```
SELECT "simple_jobs".* FROM "simple_jobs" WHERE "simple_jobs"."finished_at" IS NULL ORDER BY "simple_jobs"."id"
```

Just Add More Tables™

... to keep the polling table small

```
  create_table "simple_job_ready_executions", force: :cascade do |t|
    t.datetime "created_at"
    t.bigint "job_id", null: false
    t.index ["job_id"], name: "index_simple_job_ready_executions_on_job_id", unique: true
  end
```

```
module SimpleJob
  class ReadyExecution < ApplicationRecord
    self.table_name = "simple_job_ready_executions"

    belongs_to :job, class_name: "SimpleJob::Job"
  end
end
```

```
SELECT "simple_job_ready_executions".* FROM "simple_job_ready_executions"
```

GoodJob 📢

```
class Worker
  def start
    connection.execute("LISTEN good_job")

    loop do
      connection.wait_for_notify(timeout: 1) do |channel, payload|
        job = JSON.parse(payload)
        execute(job)
      end
    end
  end
end

def enqueue(job)
  Job.create!(job)
  connection.execute("NOTIFY good_job, job.to_json)
end
```

Sidekiq ⚡

```ruby
class Worker
  def poll
    loop do
      queue, job = redis.brpop("queue:default", timeout: 2)

      execute(job) if job
    end
  end
end

def enqueue(job)
  redis.lpush("queue:default", job.to_json)
end
```

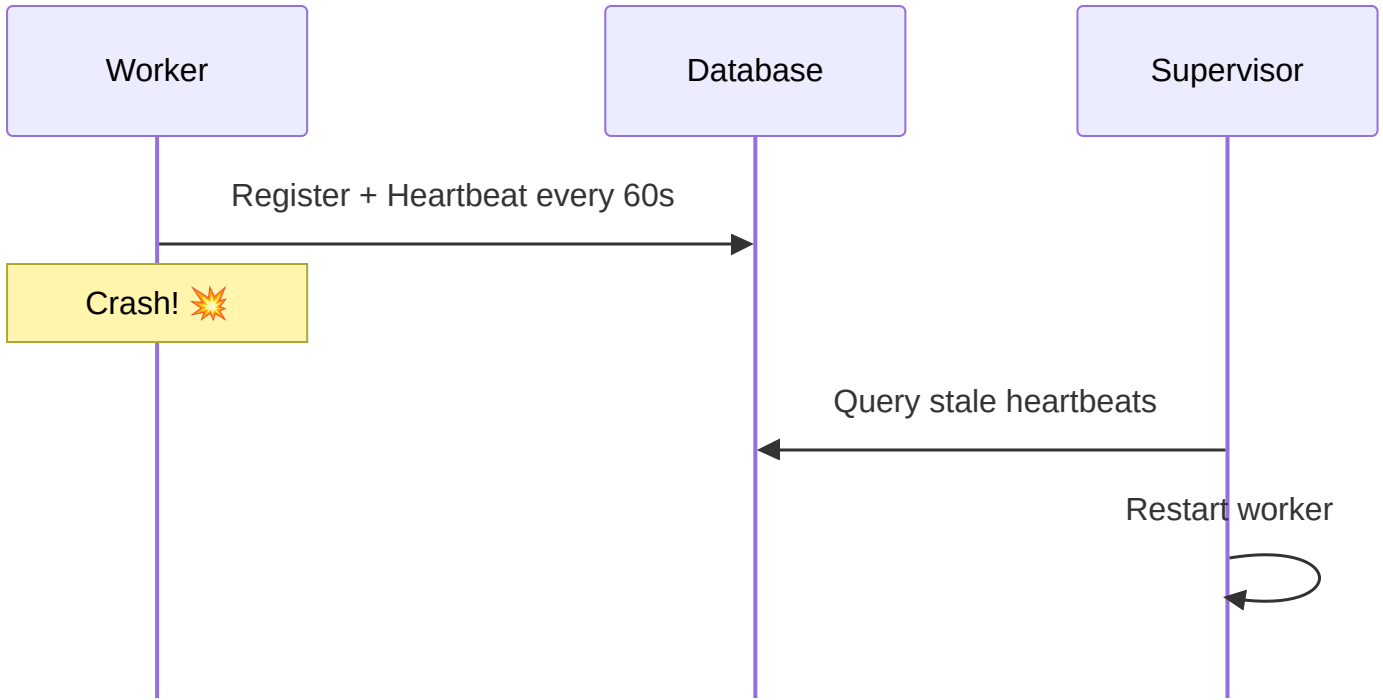
What If Workers Die? 🪦

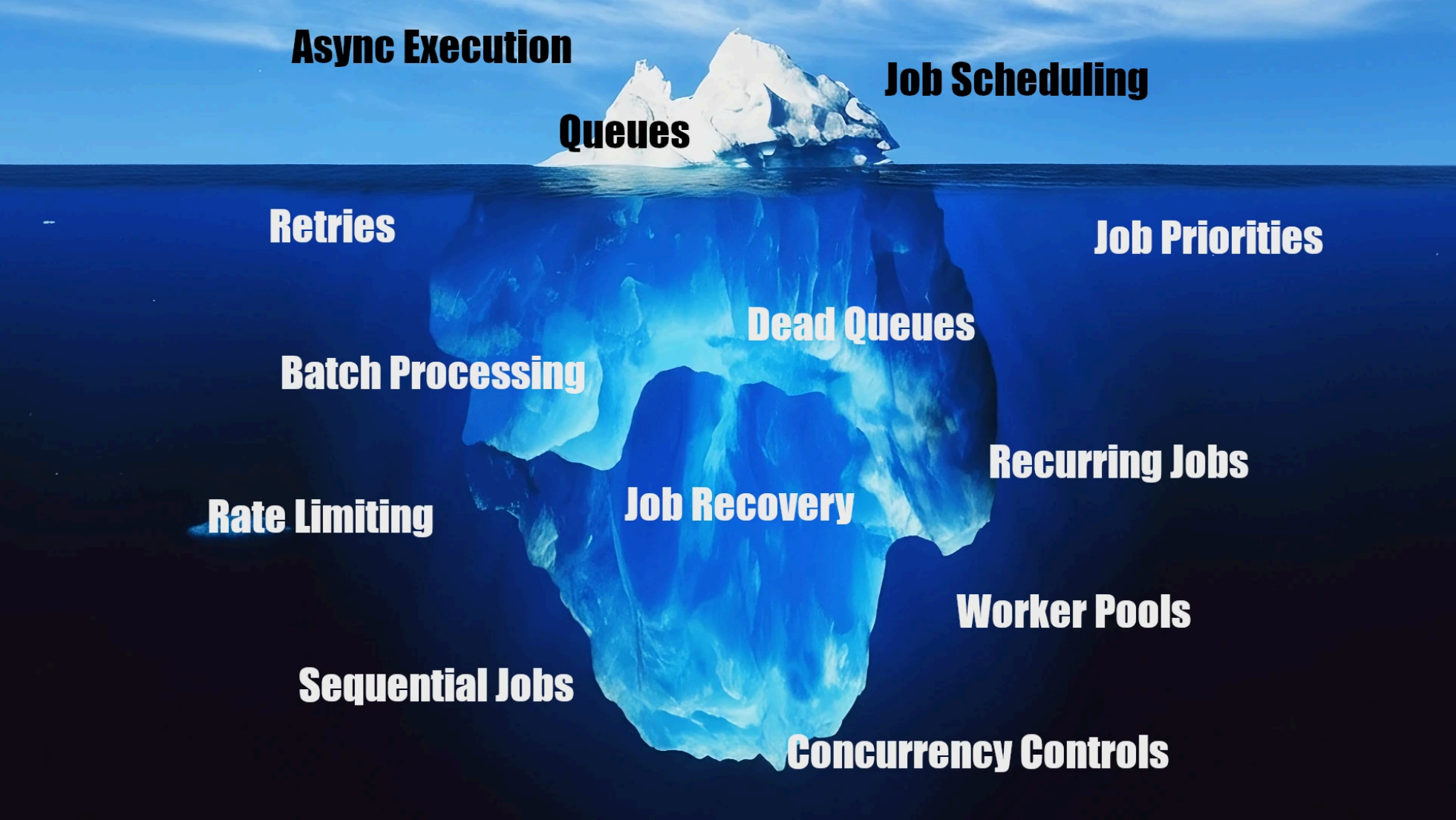
Processes die all the time. Sometimes with grace, sometimes quite unexpectedly.

```ruby
ActiveRecord::Base.transaction do
  execution.destroy!
  # 💥 Worker dies right here 💥
  execute(job)
  job.update!(finished_at: Time.current)
end
```

- Solid Queue: Use a claimed state! 👌
- GoodJob: Use Postgres Advisory Locks 📣
- Sidekiq: Buy Sidekiq Pro 🤷

Resurrecting Dead Workers 🧟




Async Execution
Job Scheduling
Queues
Retries
Job Priorities
Dead Queues
Batch Processing
Recurring Jobs
Job Recovery
Rate Limiting
Worker Pools
Sequential Jobs
Concurrency Controls

Vienna.rb #66

December 3rd, 2025

Paweł Strzałkowski

AI Interface in 5 Minutes -
Model Context Protocol on Rails

Rosa Gutiérrez

Machines that talk about themselves

🎄Christmas Cookie Exchange 🎄

It's Christmas! Bring some cookies to share with your fellow Rubyists!

Hosted by SENTRY

Thank you!

Sources & Resources

- Solid Queue Internals and Externals
- How does Sidekiq really work?
- Ben Sheldon’s Blog
- Solid Queue for Ruby on Rails
- Solid Queue Source
- Sidekiq Source
- GoodJob Source