Cheatsheet to help you decide which fix for the last-comment n+1 problem is the best for you

(… based on my opinion 🤫 … but based on benchmarks 😀)

Fetched posts	Comments per post	Requests per second	Recomendation
20	5	Less than 300	Sorted comments
20	5	More than 300	Feed looping, Fragment caching
20	10	Less than 200	Sorted comments
20	10	More than 200	Feed looping, Fragment caching
20	50	Less than 80	Sorted comments
20	50	More than 80	Feed looping, Fragment caching
20	100	Less than 700	Feed looping, Fragment caching
20	1000	Less than 200	Feed looping, Fragment caching

"Sorted comments"

```
class Post
  has_many :sorted_comments, -> { order(:created_at) }

  def latest_comment
    sorted_comments.last
  end
end

Post.includes(:sorted_comments).each do |post|
  puts post.latest_comment
end
```

"Fragment caching"

```
class Post < ActiveRecord::Base
  has_many :comments

  def latest_comment
    comments.order(:id).last
  end
end

class Comment < ActiveRecord::Base
  belongs_to :post
end
```

In your view…

```
<% #posts/index.html.erb %>
<%= render partial: 'posts/post', collection: @posts,
  cached: true %>

<% #posts/_post.html.erb %>
<% cache post %>
  <%= post.title %>
  <%= post.latest_comment.body %>
<% end %>
```

"Feed looping" through the latest comment for each post

```
class Post < ActiveRecord::Base
  has_many :comments
end

class Comment < ActiveRecord::Base
  belongs_to :post

  def self.latest_comments_for_posts
    latest_comments_ids = select("max(id)").group(:post_id)
    where(id: latest_comments_ids)
  end
end

class Feed
  def posts
    posts = Post.all
    comments = Comment.latest_comments_for_posts.group_by(&:post_id)
    posts.map { |post| FeedPost.new(post, comments[post.id]&.first) }
  end
end

class FeedPost
  attr_reader :latest_comment

  def initialize(post, latest_comment)
    @post = post
    @latest_comment = latest_comment
  end
end

feed = Feed.new

feed.posts.map do |post|
  puts post.latest_comment.body
end
```