WELCOME...

THANKS FOR JOINING THE TEMPORAL LONDON MEETUP!

AGENDA

01: **WELCOME**
GRAHAM PYMM - TEMPORAL TECHNOLOGIES

02: **IMPLEMENTING BATCH JOBS WITH TEMPORAL**
MAXIM FATEEV, CEO - TEMPORAL TECHNOLOGIES

03: **SCALING LONG RUNNING OPERATIONS AT SNYK**
IMRAN BOHORAN, NEAL MORRIS, AND TAWHID HANNAN - SNYK

04: **FROM POC TO PRODUCTION WITH TEMPORAL**
TIHOMIR SURDILOVIC - TEMPORAL TECHNOLOGIES

MULTI-REGION AVAILABILITY HAS COME TO TEMPORAL CLOUD

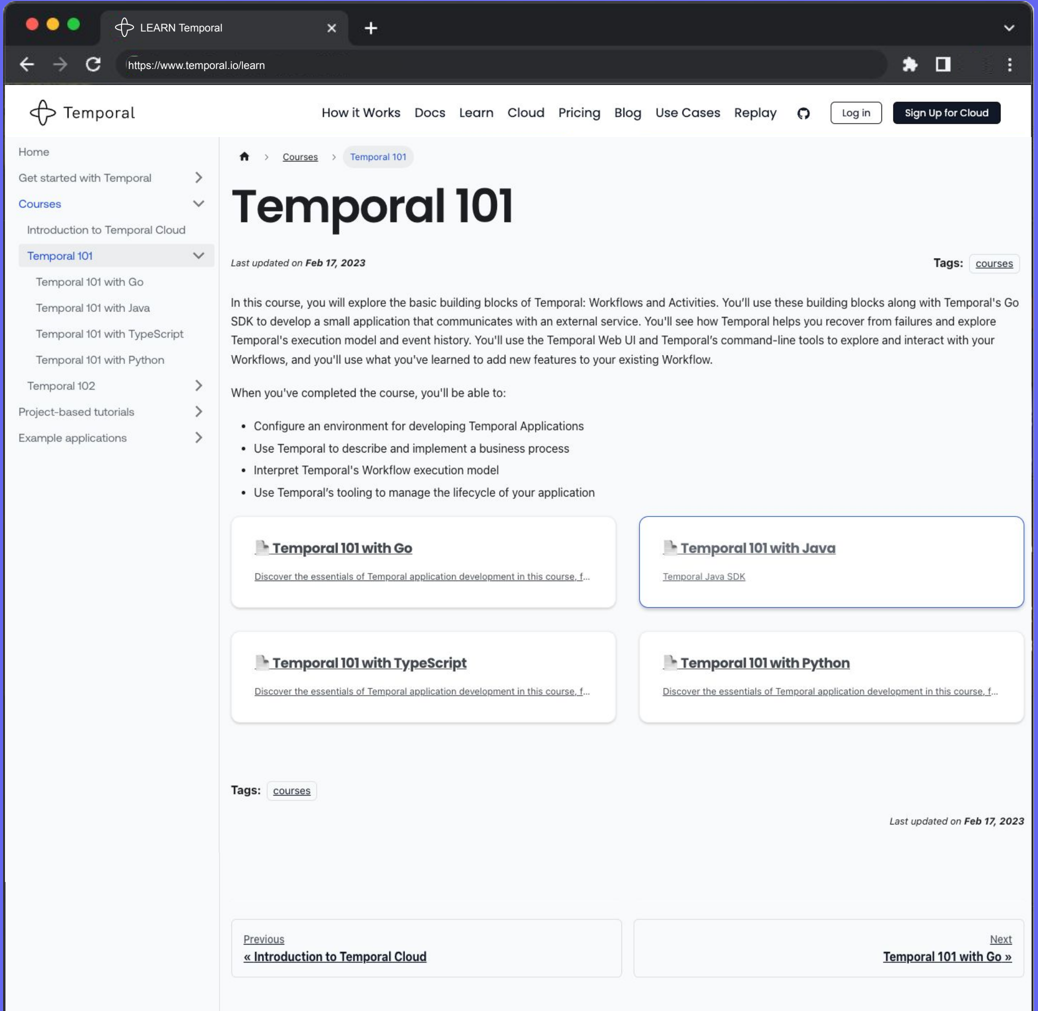
Run thousands of actions per second across regions without breaking sweat.

LEARN

Self-paced online courses that provide in-depth hands-on learning experiences.

- Temporal 101
- Temporal 102
- Intro to Temporal Cloud

https://learn.temporal.io/courses


LEARN Temporal

https://www.temporal.io/learn

Temporal

How it Works Docs Learn Cloud Pricing Blog Use Cases Replay Log in Sign Up for Cloud

- Home
- Get started with Temporal
- Courses
  - Introduction to Temporal Cloud
  - Temporal 101
    - Temporal 101 with Go
    - Temporal 101 with Java
    - Temporal 101 with TypeScript
    - Temporal 101 with Python
  - Temporal 102
- Project-based tutorials
- Example applications

Courses › Temporal 101

Temporal 101

Last updated on ***Feb 17, 2023***

Tags: courses

In this course, you will explore the basic building blocks of Temporal: Workflows and Activities. You'll use these building blocks along with Temporal's Go SDK to develop a small application that communicates with an external service. You'll see how Temporal helps you recover from failures and explore Temporal's execution model and event history. You'll use the Temporal Web UI and Temporal's command-line tools to explore and interact with your Workflows, and you'll use what you've learned to add new features to your existing Workflow.

When you've completed the course, you'll be able to:

- Configure an environment for developing Temporal Applications
- Use Temporal to describe and implement a business process
- Interpret Temporal's Workflow execution model
- Use Temporal's tooling to manage the lifecycle of your application

Temporal 101 with Go
Discover the essentials of Temporal application development in this course, f...

Temporal 101 with Java
Temporal Java SDK

Temporal 101 with TypeScript
Discover the essentials of Temporal application development in this course, f...

Temporal 101 with Python
Discover the essentials of Temporal application development in this course, f...

Tags: courses

Last updated on ***Feb 17, 2023***

Previous
« Introduction to Temporal Cloud

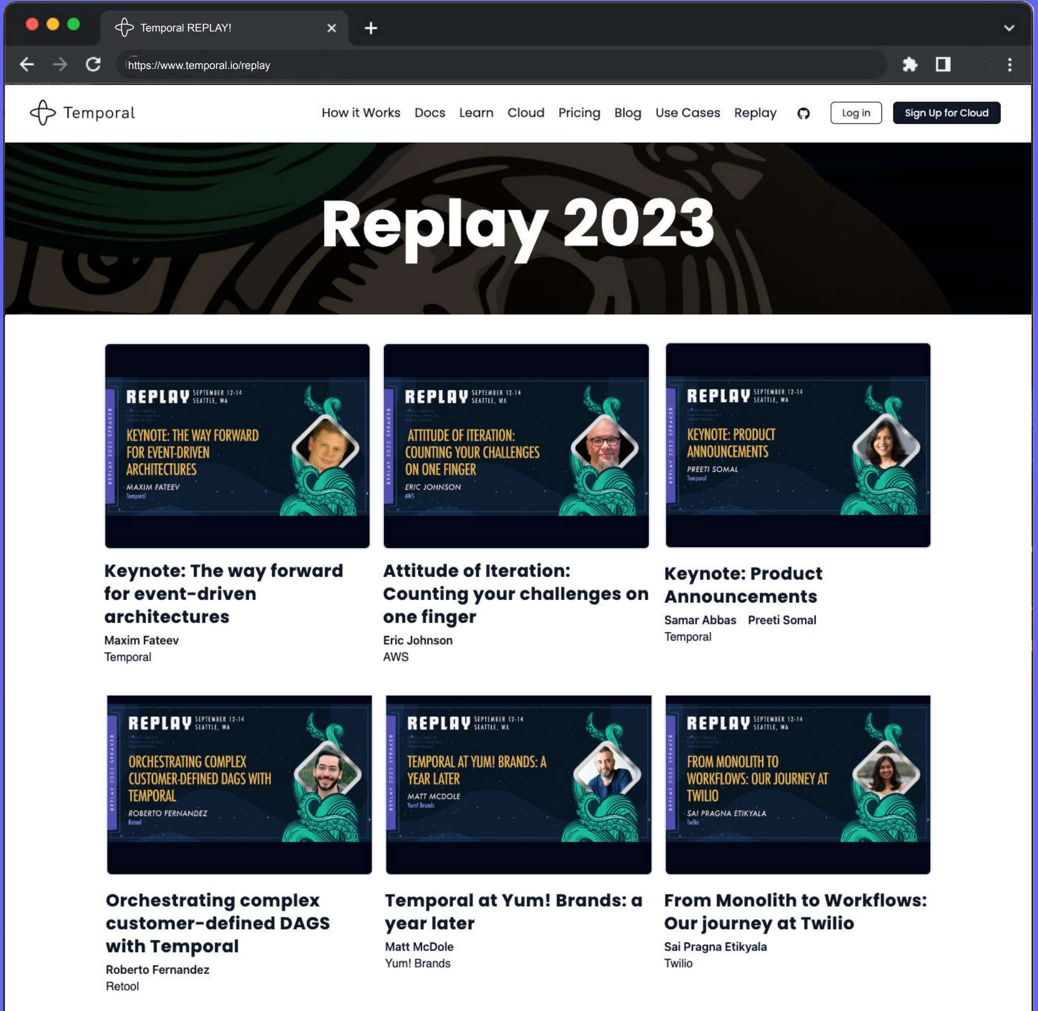
Next
Temporal 101 with Go »

REPLAY

Last month, we gathered Temporal experts from around the world to share best practices and how-to's.

You can check out **ALL** the videos at the Replay site:

https://temporal.io/replay/videos



Keynote: The way forward for event-driven architectures
Maxim Fateev
Temporal

Attitude of Iteration: Counting your challenges on one finger
Eric Johnson
AWS

Keynote: Product Announcements
Samar Abbas Preeti Somal
Temporal

Orchestrating complex customer-defined DAGS with Temporal
Roberto Fernandez
Retool

Temporal at Yum! Brands: a year later
Matt McDole
Yum! Brands

From Monolith to Workflows: Our journey at Twilio
Sai Pragna Etikyala
Twilio

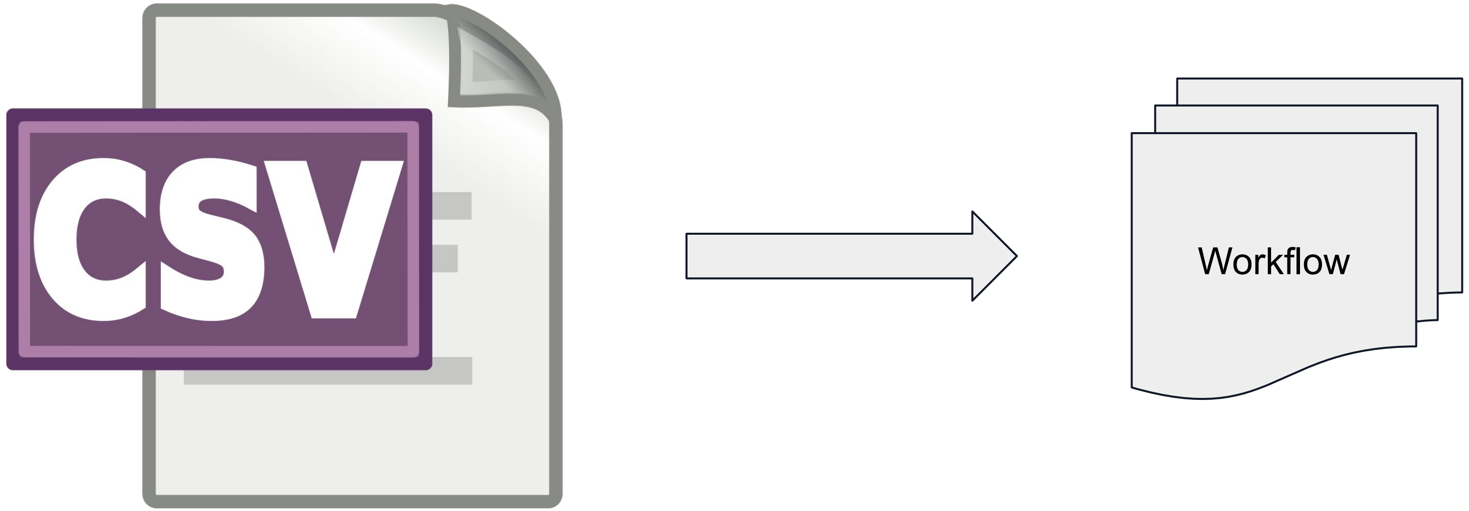
Implementing Batch Jobs

With Temporal

Outline

- Definition of batch
- Naive Approach
- Heartbeating Activity
- Iterator Workflow
- Sliding Window

Batch Job



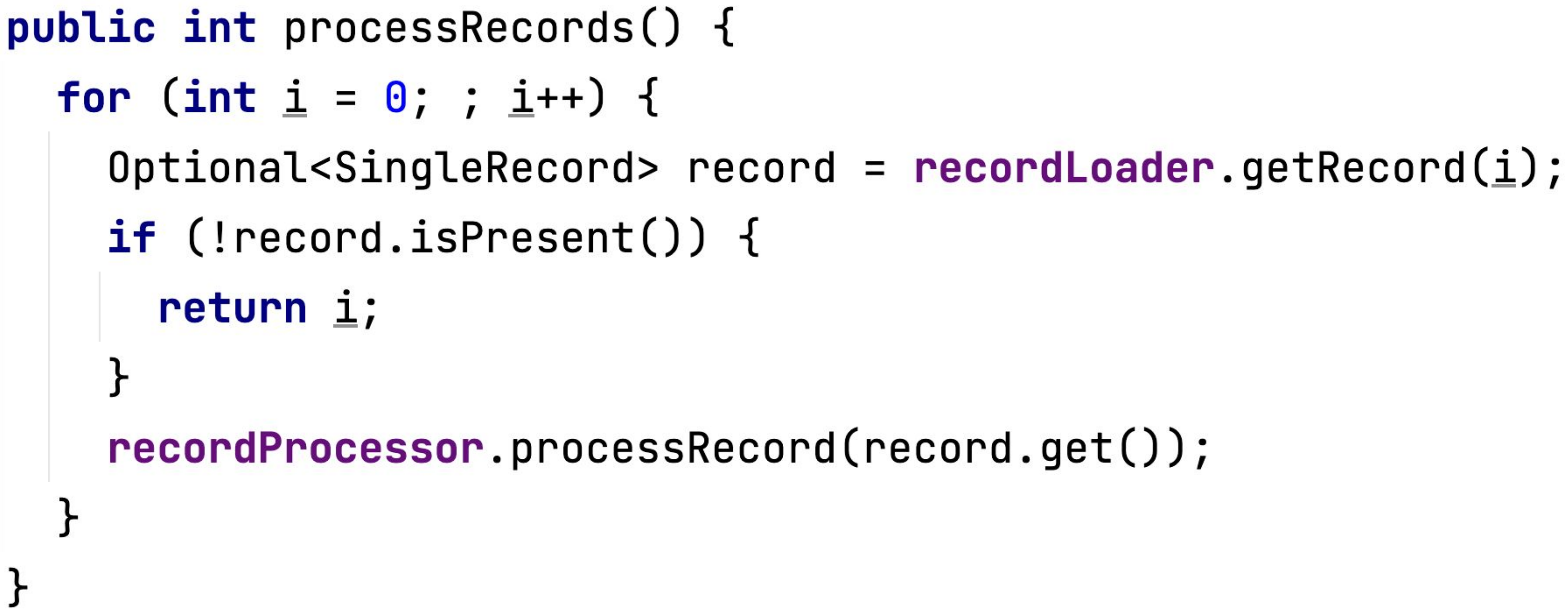
Naive

```java
public int processBatch() {
  List<SingleRecord> records = recordLoader.getRecords();
  for (SingleRecord record : records) {
    RecordProcessorWorkflow processor =
        Workflow.newChildWorkflowStub(RecordProcessorWorkflow.class);
    processor.processRecord(record);
  }
  return records.size();
}
```

Naive Implementation

- Drawback
  - Batch size limit
- Advantage
  - Simplicity
- Improvements
  - Parallel record processing

Activity



```java
public int processRecords() {
  for (int i = 0; ; i++) {
    Optional<SingleRecord> record = recordLoader.getRecord(i);
    if (!record.isPresent()) {
      return i;
    }
    recordProcessor.processRecord(record.get());
  }
}
```

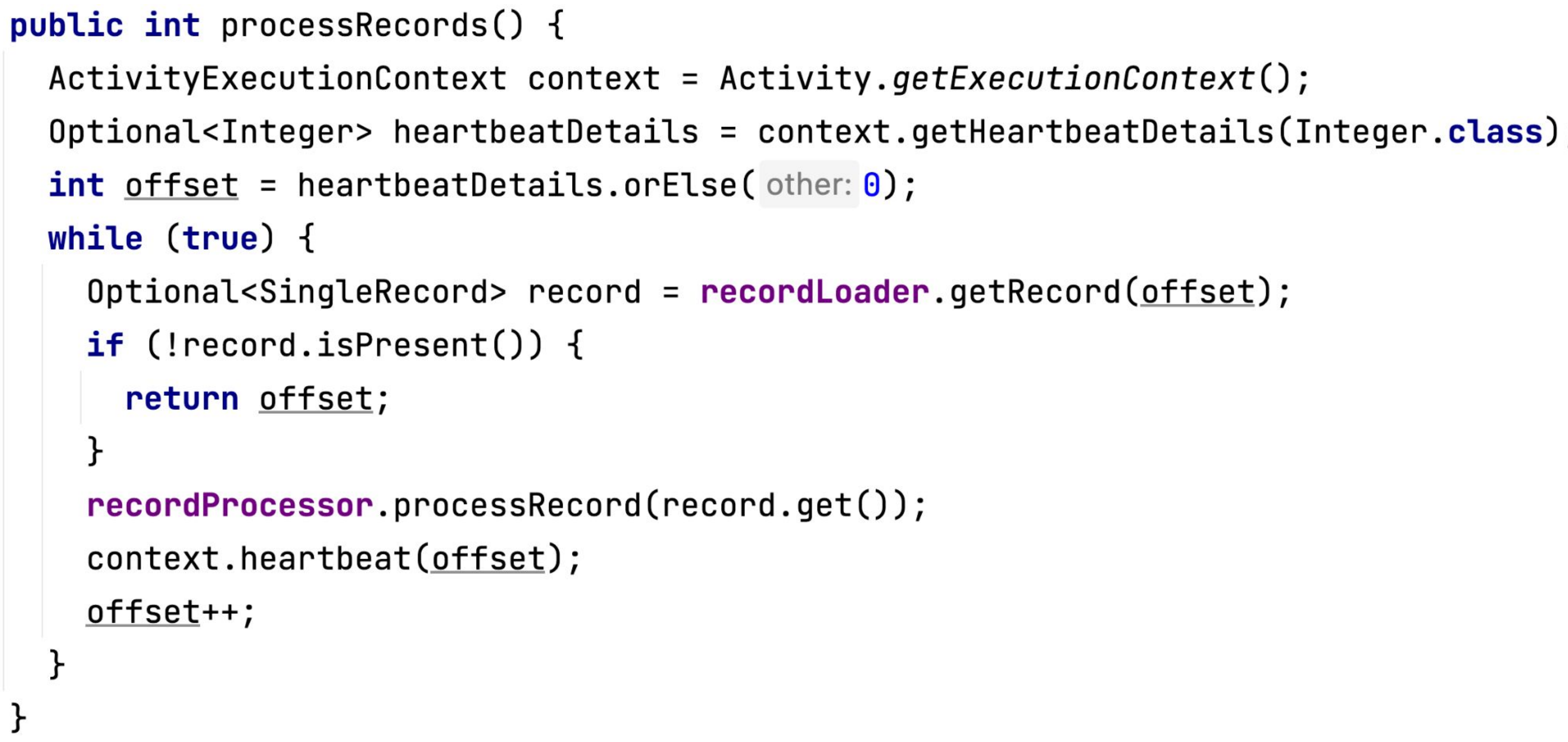
Activity

- Drawbacks
  - Only fast record processing
  - No poison pill support
  - Reprocessing of the whole dataset on retries
- Advantage
  - Low resource utilization
- Improvements
  - Parallel record processing
  - Parallel activities

Activity

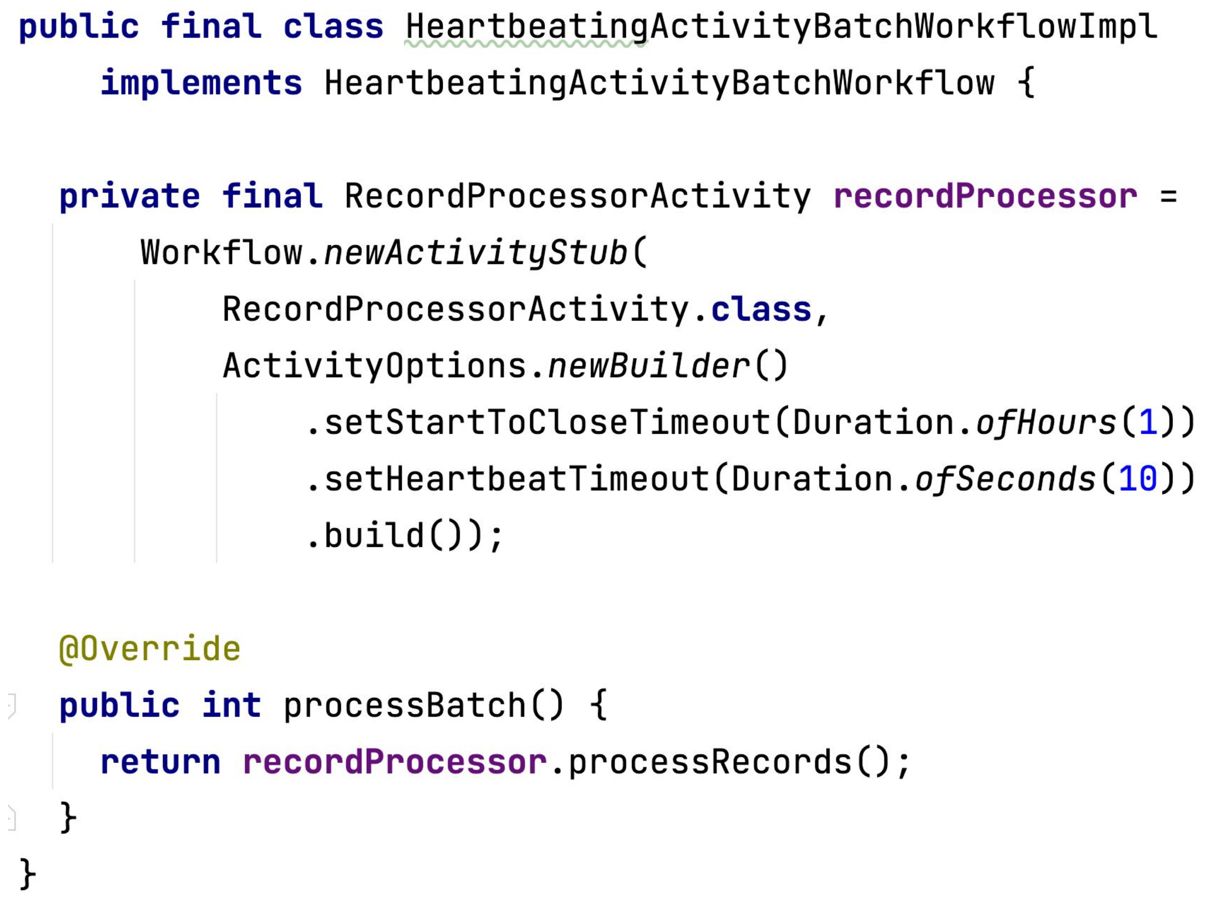
- Drawbacks
  - Only fast record processing
  - No poison pill support
  - **Reprocessing of the whole dataset on retries**
- Advantage
  - Low resource utilization
- Improvements
  - Parallel record processing
  - Parallel activities
  - Activity heartbeating

Heartbeating Activity



```java
public int processRecords() {
  ActivityExecutionContext context = Activity.getExecutionContext();
  Optional<Integer> heartbeatDetails = context.getHeartbeatDetails(Integer.class)
  int offset = heartbeatDetails.orElse( other: 0);
  while (true) {
    Optional<SingleRecord> record = recordLoader.getRecord(offset);
    if (!record.isPresent()) {
      return offset;
    }
    recordProcessor.processRecord(record.get());
    context.heartbeat(offset);
    offset++;
  }
}
```

Heartbeating Activity Workflow



```java
public final class HeartbeatingActivityBatchWorkflowImpl
    implements HeartbeatingActivityBatchWorkflow {

  private final RecordProcessorActivity recordProcessor =
      Workflow.newActivityStub(
          RecordProcessorActivity.class,
          ActivityOptions.newBuilder()
              .setStartToCloseTimeout(Duration.ofHours(1))
              .setHeartbeatTimeout(Duration.ofSeconds(10))
              .build());

  @Override
  public int processBatch() {
    return recordProcessor.processRecords();
  }
}
```

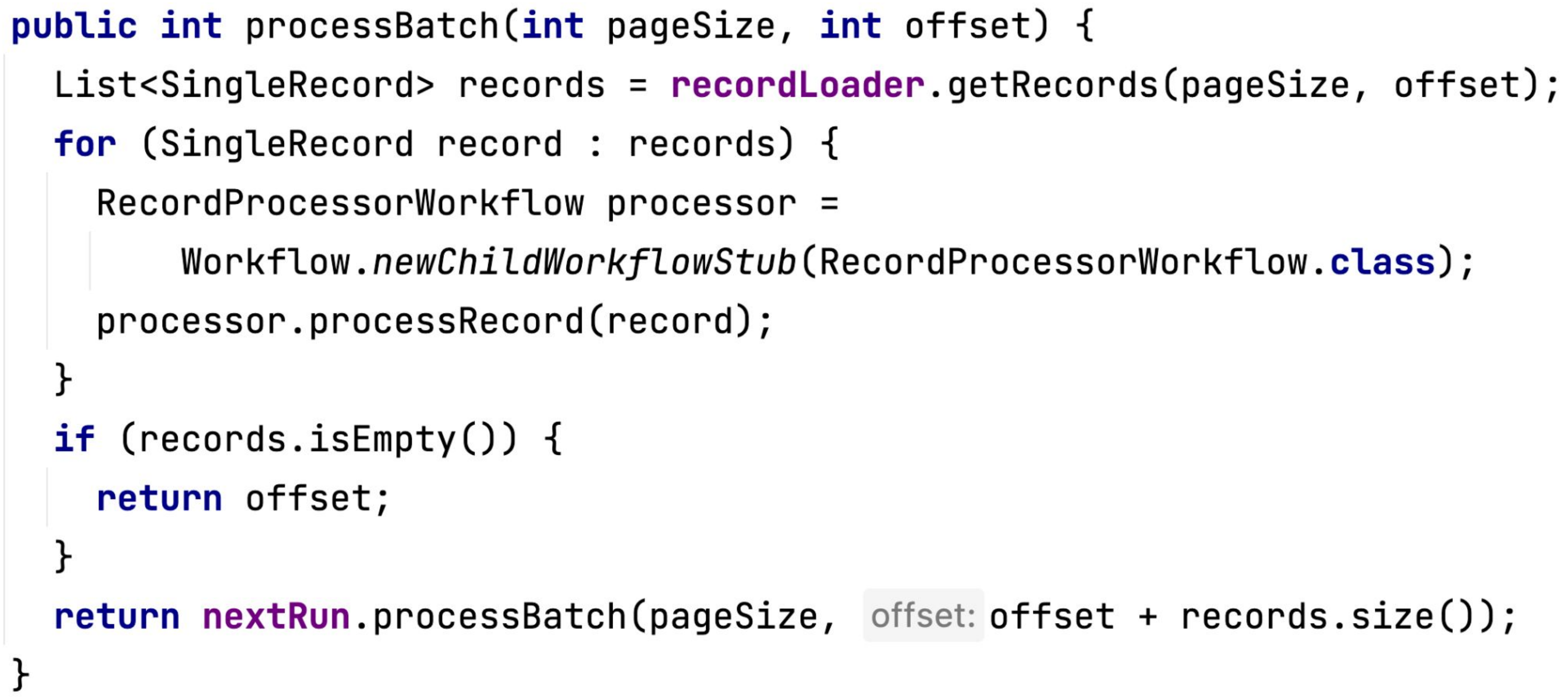
Heartbeating Activity

- Drawbacks
  - Only fast record processing
  - No poison pill support
- Advantage
  - Low resource utilization
- Improvements
  - Parallel record processing
  - Parallel activities

Iterator Workflow

- Load a page of records using an activity
- Process each record by a child workflow
- Call continue-as-new to process the rest of records

Iterator Workflow



```
public int processBatch(int pageSize, int offset) {
  List<SingleRecord> records = recordLoader.getRecords(pageSize, offset);
  for (SingleRecord record : records) {
    RecordProcessorWorkflow processor =
        Workflow.newChildWorkflowStub(RecordProcessorWorkflow.class);
    processor.processRecord(record);
  }
  if (records.isEmpty()) {
    return offset;
  }
  return nextRun.processBatch(pageSize, offset: offset + records.size());
}
```

Iterator Workflow

- Drawbacks
  - The throughput depends on the slowest workflow in each iteration
  - Creates spiky resource utilization pattern
- Advantage
  - Unlimited dataset size
- Improvements
  - Parallel record processing
  - Parallel iterator workflows

Sliding Window

- Starts a predefined number (window size) of child workflows
- Parent calls continue-as-new
- A child upon processing a record signals the parent
- The parent starts the next child workflow upon receiving the signal

Sliding Window

- Drawbacks
  - Complexity
- Advantage
  - Even resource utilization
- Improvements
  - Parallel sliding windows

Recap

- Naive
- Heartbeating activity
- Iterator
- Sliding Window

https://github.com/temporalio/samples-java/tree/main/core/src/main/java/io/temporal/samples/batch

https://github.com/temporalio/samples-go/tree/main/batch-sliding-window

Questions

Temporal

AGENDA

03: **SCALING LONG RUNNING OPERATIONS AT SNYK**

IMRAN BOHORAN, NEAL MORRIS, AND TAWHID HANNAN - SNYK

AGENDA

04: **FROM POC TO PRODUCTION WITH TEMPORAL**

TIHOMIR SURDILOVIC - TEMPORAL TECHNOLOGIES

About me

Design

Code Review Worker Tuning

Best Practices

Performance Use Case Observability

Your journey is yours.

Put yourself in best position for success.

Demo

Test
Observe
Detect
Understand
Improve

THANK YOU...

QUESTIONS?