Building Assembly Lines in Temporal

Link to Talk & Sample Code

JURISTAT

Introductions

Robert Ward

- Co-founder / Principal Architect @ Juristat
- Co-founder @ Arch Reactor Hackerspace

Juristat

- Started in 2012 as an analytics company
- Expanded in 2019 to Workflow Automation
- Other company details?

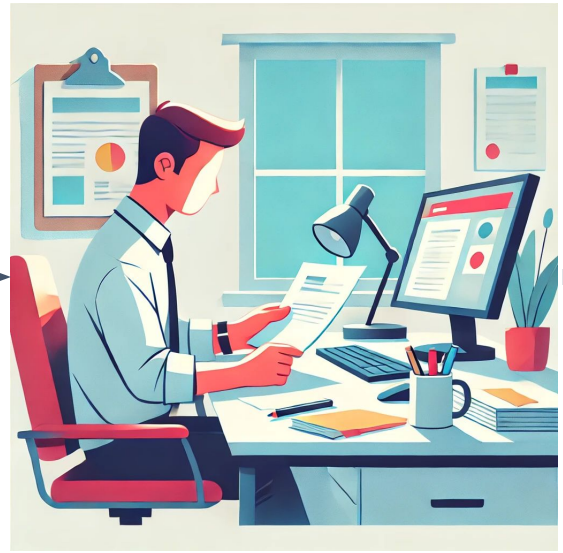
What is Workflow Automation?

Helping patent attorneys work better and faster

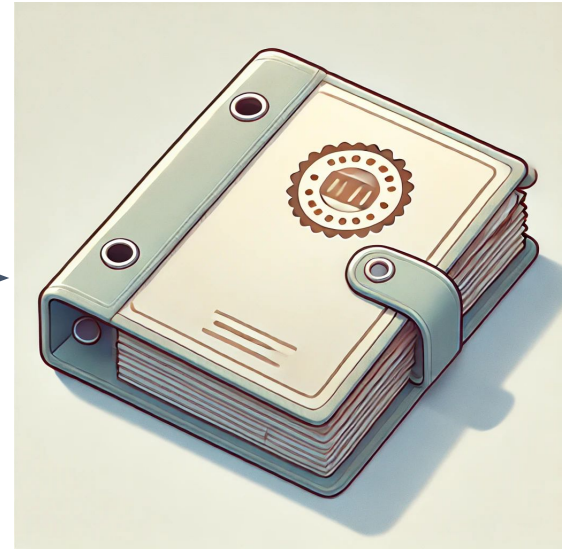
- Collect research materials
- Prepare document boilerplate
- Annotate documents
- Handle basic communications
- Packets definition

The Problem

Inputs and prep work
from automated systems

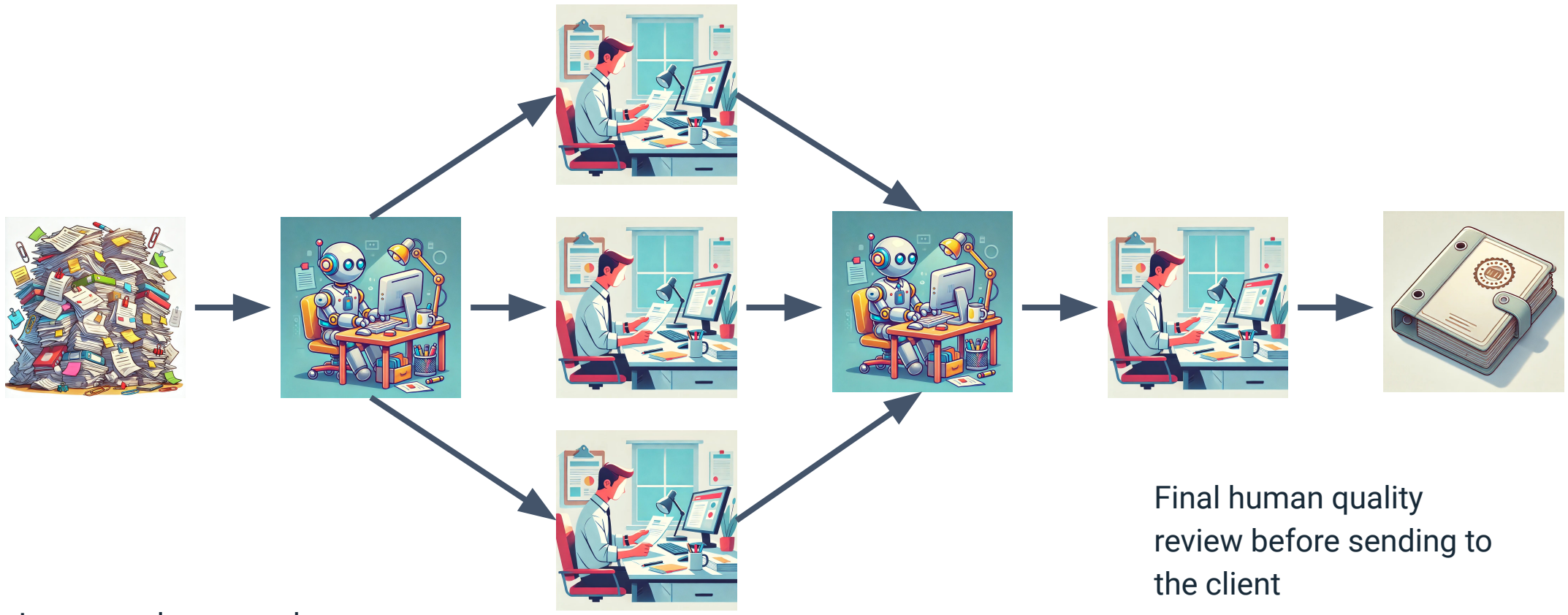


Highly trained analyst
doing end-to-end work



Finished product ready to
send to a client

The Solution: Assembly Lines



Inputs and prep work from automated systems

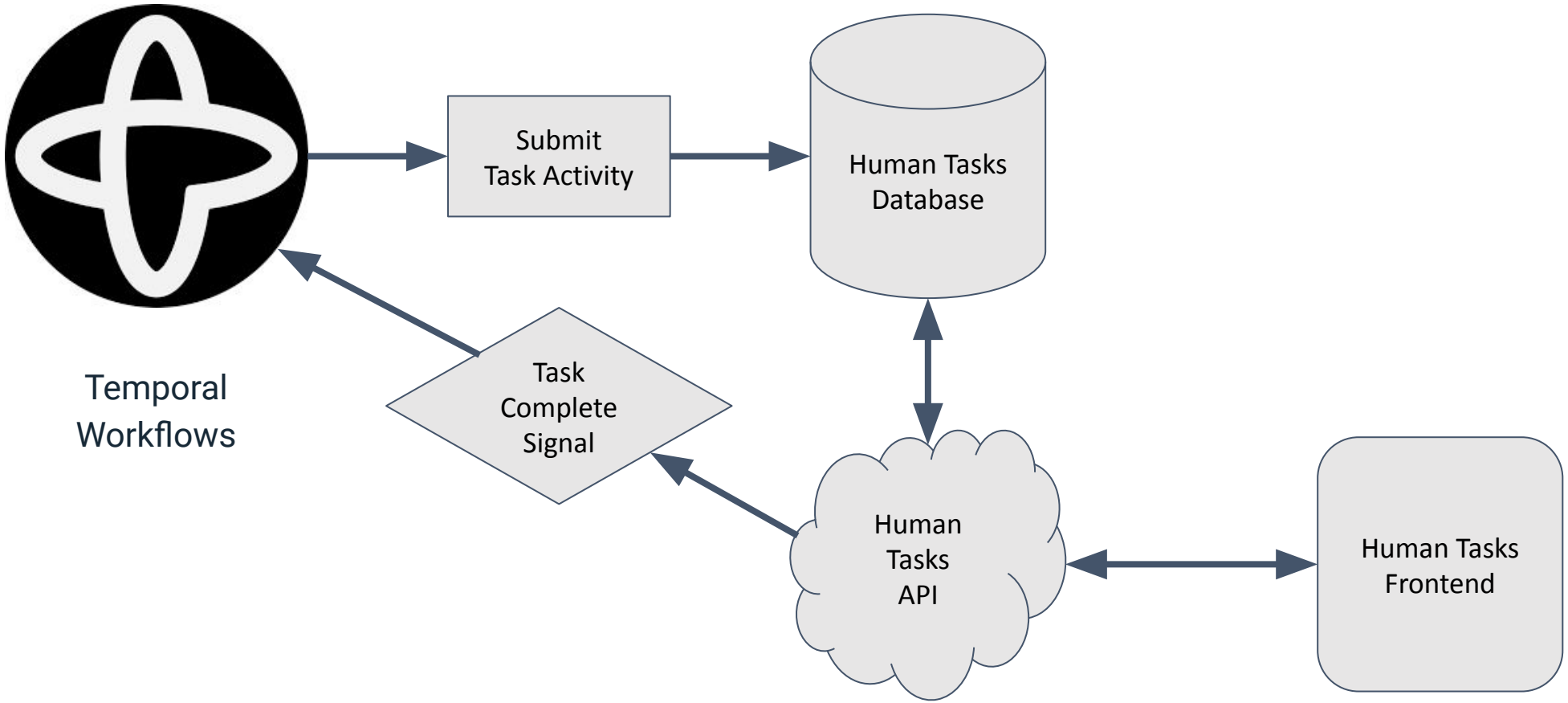
Mixed workflow of small human tasks and automated systems

Final human quality review before sending to the client

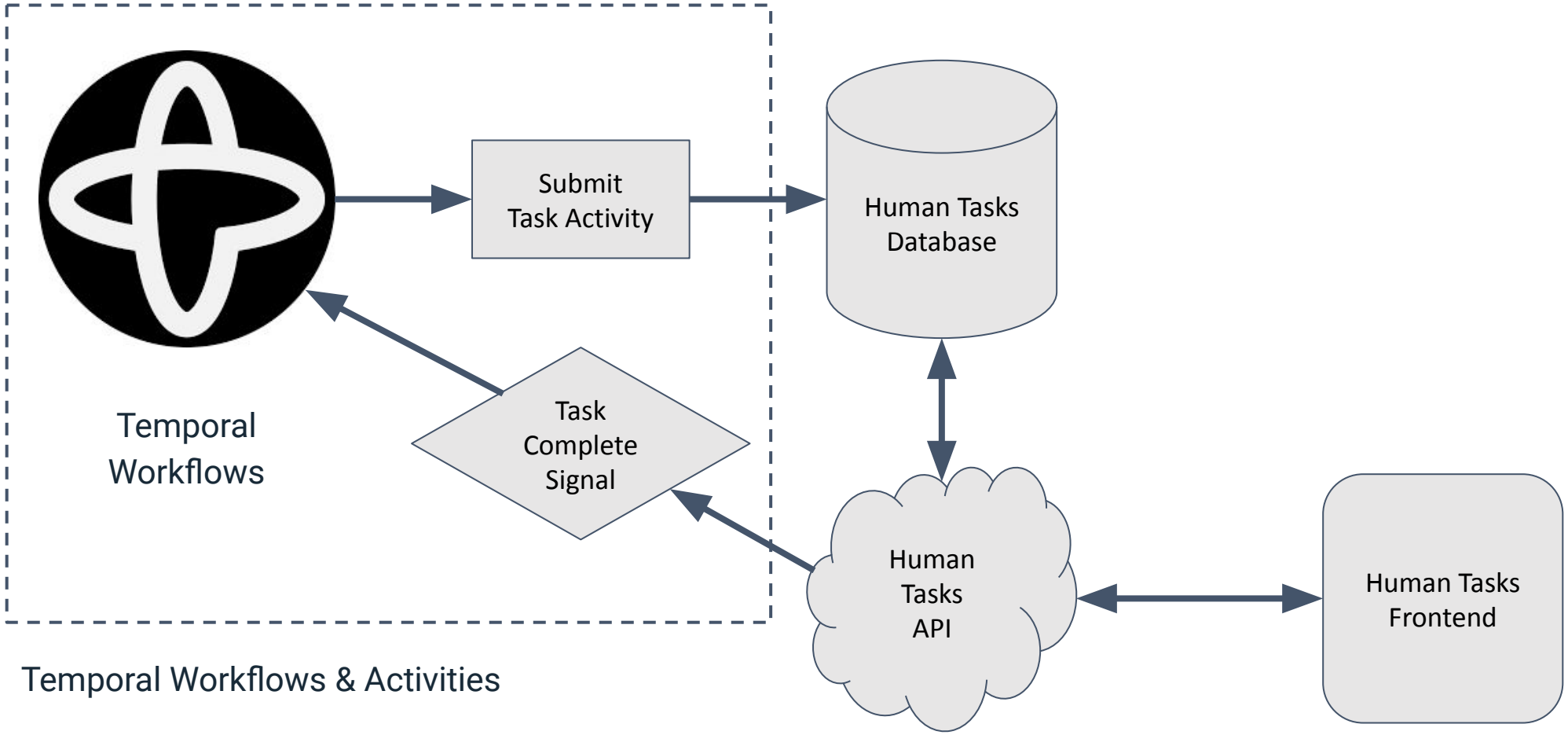
Solution Goals

- Define the whole workflow process end-to-end in code
- Make it easy to call a human task
- Small reusable human tasks
- Parallelize work where possible
- Handle deduplication
- Don’t re-do work

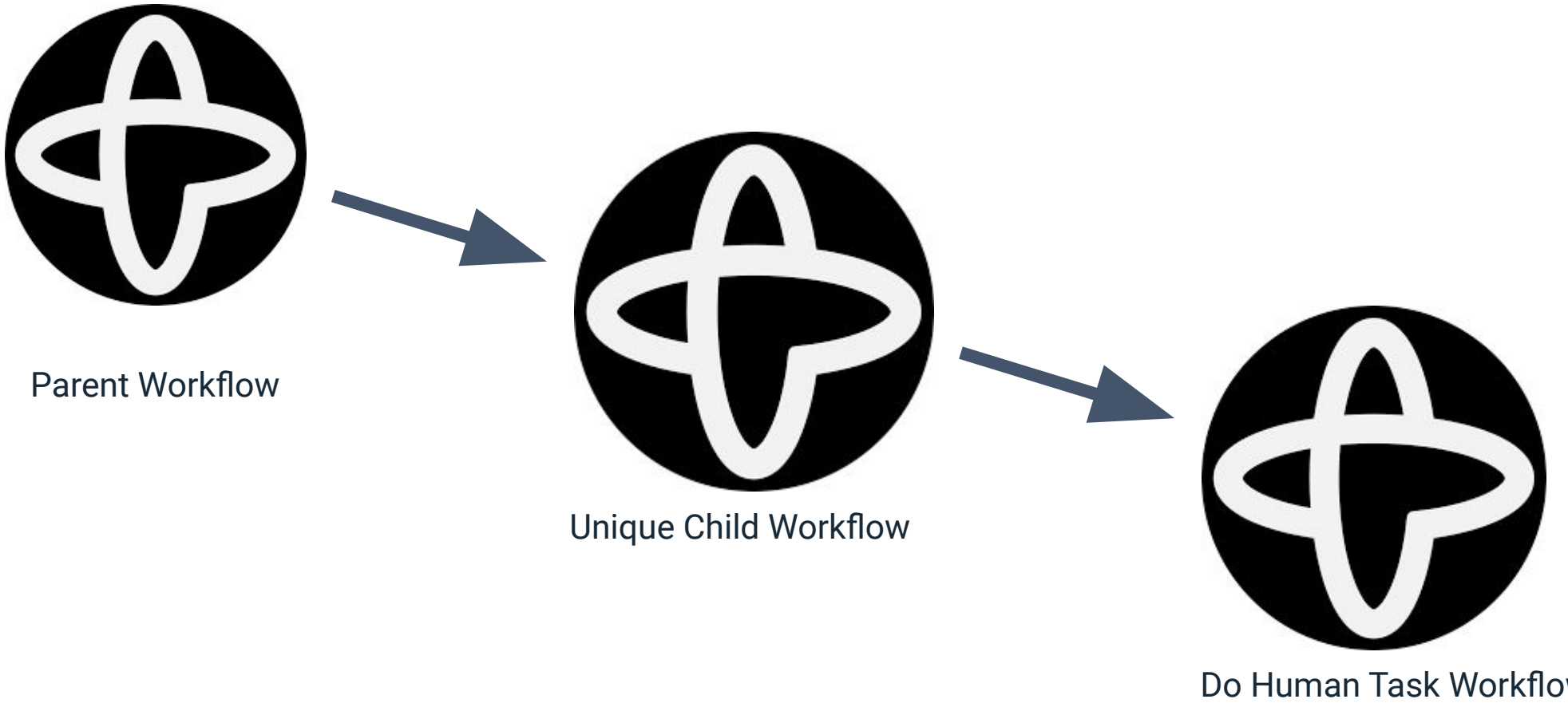
High Level Design



High Level Design



Temporal Workflows



Temporal Workflows

Parent Workflow

- Where your actual workflow code lives
- You need the result of a human task
- Can call a human task almost like a function

Parent Workflow

```
export async function myWorkflow(input: string): Promise<void> {
 const workToHandle = await doActivity(input)

 const humanTaskResult = await executeChild(humanTask, {
   workflowId: `humanTask-${workflowInfo().workflowId}-${uuid4()}`,
   workflowRunTimeout: '7d',
   args: [workToHandle],
 })

 await doAnotherActivity(humanTaskResult)
}
```

Temporal Workflows

Unique Child Workflow

- Generate workflow IDs with a unique ID
- Can be called from multiple workflows
- Handles deduplication and routing of results

Unique Child Workflow

```
export async function humanTask(args: TaskInput): Promise<TaskOutput> {
 let result = undefined
 setHandler(humanTaskCompletedSignal, (paylaod) => {
   result = paylaod
 })

 await signalWithStartHumanTask(args)

 await condition(() => result !== undefined)

 if (result.success) return result.output
 else throw result.error
}
```

Signal With Start Activity

```
export async function signalWithStartHumanTask(args: TaskInput): Promise<void>{
 const workflowId = await getDeterministicWorkflowId(args)
 const ctx = await Context.current()

 await client.workflow.signalWithStart(doHumanTask, {
   workflowId,
   workflowRunTimeout: '7d',
   taskQueue: ctx.info.taskQueue,
   args: [args],
   signal: subscribeToHumanTaskCompletedSignal,
   signalArgs: [{ workflowId: ctx.info.workflowExecution.workflowId }],
 })
}
```

Temporal Workflows

- Handles the actual human task interaction
- Performs pre and post processing
- Handles memoization of task results
- Will only be called once for any given task

Do Human Task Workflow

Do Human Task Workflow

```
export async function doHumanTask(args: TaskInput): Promise<TaskOutput> {
 const subscriptions = new Set<string>()
 setHandler(subscribeToHumanTaskCompletedSignal, async ({ workflowId }) =>
   subscriptions.add(workflowId)
 )

 let result = undefined
 setHandler(externalHumanTaskCompletedSignal, (payload) => {
   result = payload
 })
 // …
}
```

Do Human Task Workflow

```
export async function doHumanTask(args: TaskInput): Promise<TaskOutput> {
 // …
 try {
   const memoizedResult = await checkForMemoizedHumanTaskResult(args)
   if (memoizedResult) return await signalSuccess(subsciptions, memoizedResult)
// …
}
```

Do Human Task Workflow

```
export async function doHumanTask(args: TaskInput): Promise<TaskOutput> {
 // …
 try {
  // …
   await validateInput(args)


   await submitHumanTaskToExternalSystem(args)


   await condition(() => result !== undefined)
  // …
```

Do Human Task Workflow

```
export async function doHumanTask(args: TaskInput): Promise<TaskOutput> {
 // …
 try {
  // …
   if (result.error) throw error

   await handlePostProcessing(result.output)

   await signalSuccess(subscriptions, result)
 } catch (error) {
   await signalError(subscriptions, error)
 }
}
```

Unique Child Workflow

```
export async function humanTask(args: TaskInput): Promise<TaskOutput> {
 let result = undefined
 setHandler(humanTaskCompletedSignal, (paylaod) => {
   result = paylaod
 })

 await signalWithStartHumanTask(args)

 await condition(() => result !== undefined)

 if (result.success) return result.output
 else throw result.error
}
```

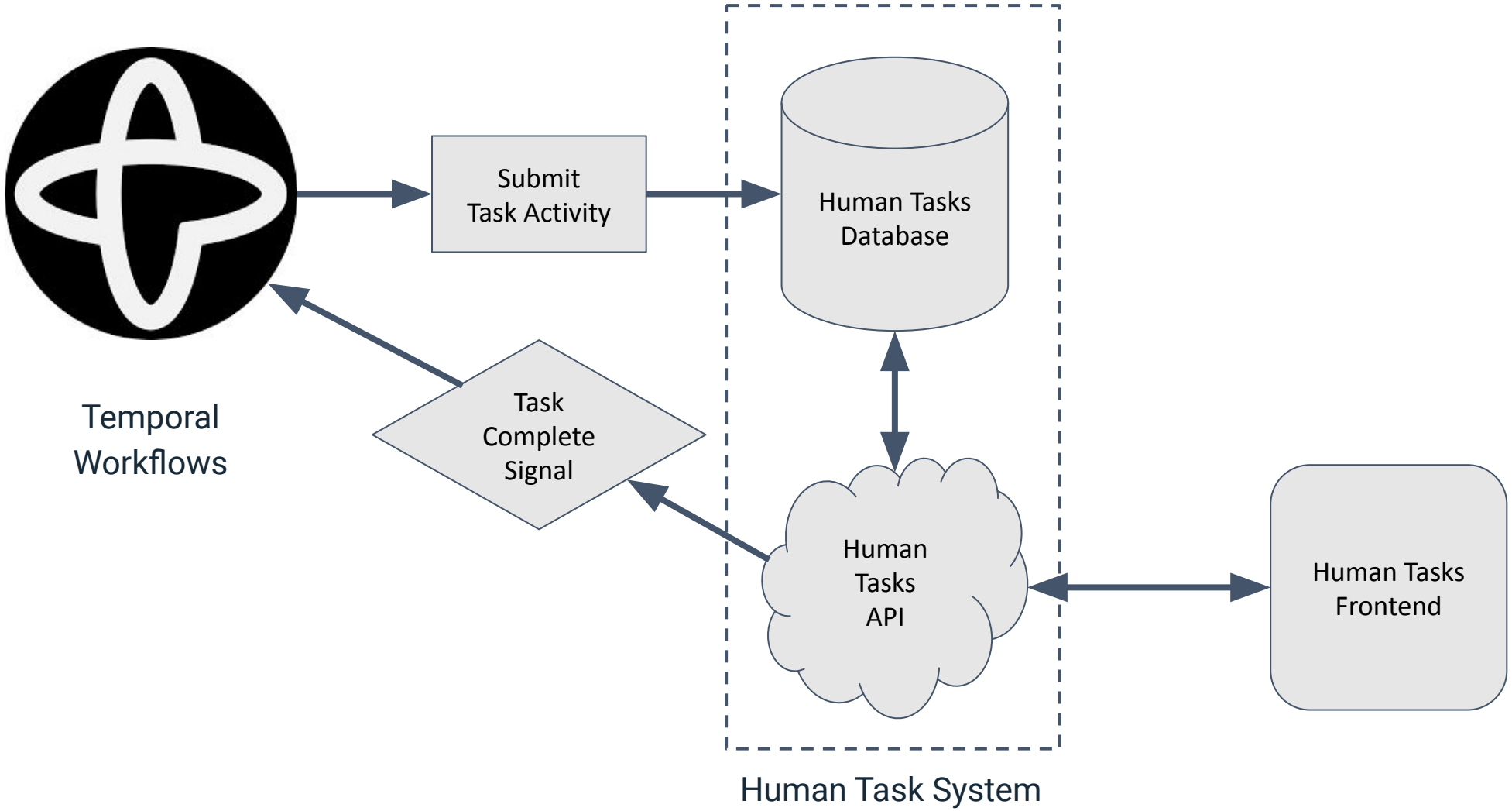
Parent Workflow

```
export async function myWorkflow(input: string): Promise<void> {
 const workToHandle = await doActivity(input)

 const humanTaskResult = await executeChild(humanTask, {
   workflowId: `humanTask-${workflowInfo().workflowId}-${uuid4()}`,
   workflowRunTimeout: '7d',
   args: [workToHandle],
 })

 await doAnotherActivity(humanTaskResult)
}
```

High Level Design



Tasks API

- Start a Task
- Heartbeat a Task
- Complete a Task

Start a Task

- In a transaction …
  - List available tasks
  - Self-assign the next one
- Return the task input

Code Sample Here

Heartbeat a Task

- At a set interval
- Mark the task as still being worked on

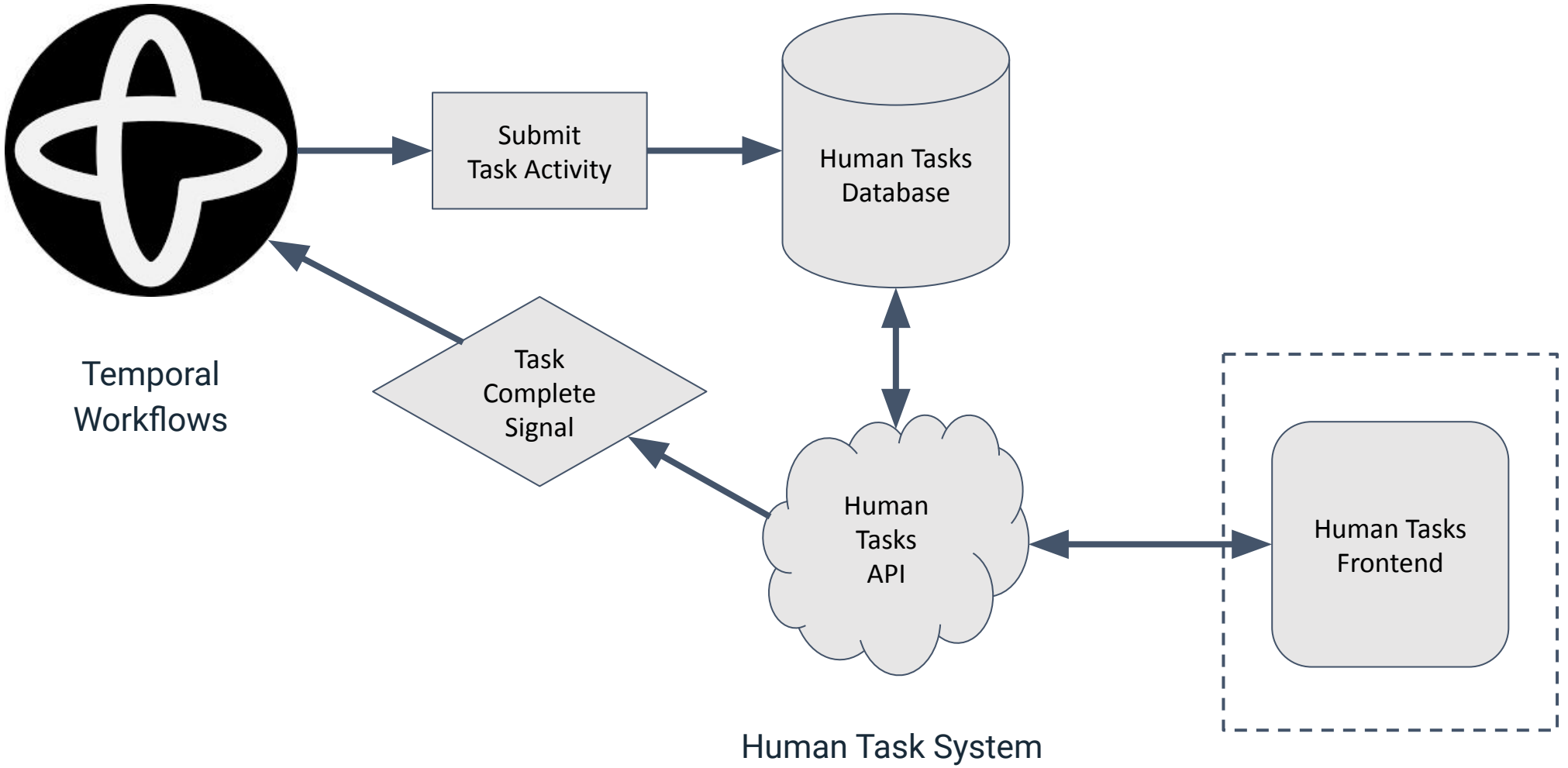
Code Sample Here

Complete a Task

- In a transaction …
  - Mark the task as complete
  - Signal the workflow with the output

Code Sample Here

High Level Design



Not even going to get into it...

- Each task will need a dedicated frontend
- Can be simple or complicated
- Need JS to heartbeat

Results?

Results!

- Generating Citation Sheets
  - Documents parsed in hours instead of days
  - Much reduced errors due to automation
  - Able to alter workflow in code instead of worker instructions

Future Projects

- Integrate into more workflows
- End to end workflow capture
- External workers
- Switch to updates for completion

Comments?
Questions?
Concerns?

https://github.com/rtward/temporal-assembly-line-talk

JURISTAT

Robert Ward
robert@juristat.com
@rtward

https://github.com/rtward/temporal-assembly-line-talk

JURISTAT