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

- Mission: Data-driven solutions to improve patent outcomes
- What if obtaining a patent was faster, easier, and more predictable?
- Started in 2012 as an Patent Analytics product
- Expanded in 2019 to Workflow Automation
- Works with ⅓ of the top 100 patent law firms, and tech giants like Taiwan Semiconductor Manufacturing and Lenovo

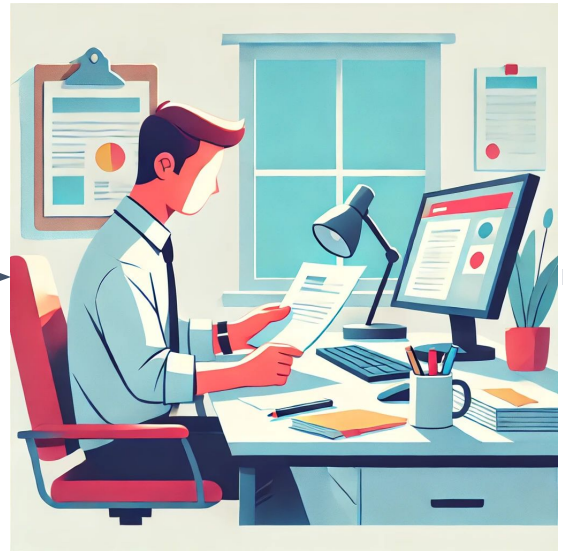
What is Workflow Automation?

Helping patent attorneys work better and faster, with fewer errors:

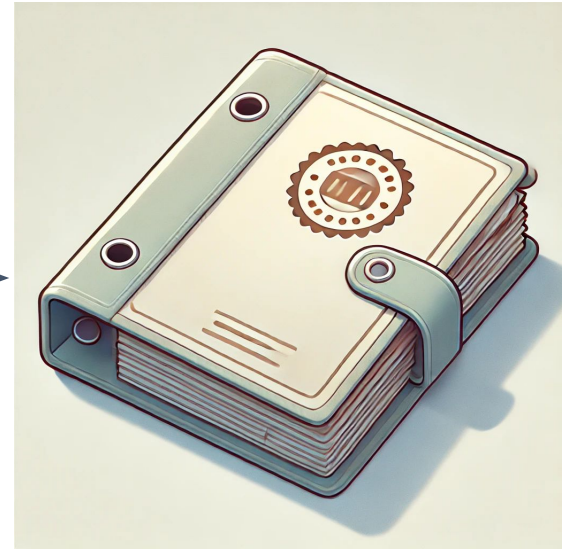
- Collect research materials
- Prepare document boilerplate
- Annotate documents
- Handle basic communications
- Minimize repetitive data entry
- Apply consistent file naming and formatting

The Problem

Inputs and prep work
from automated systems

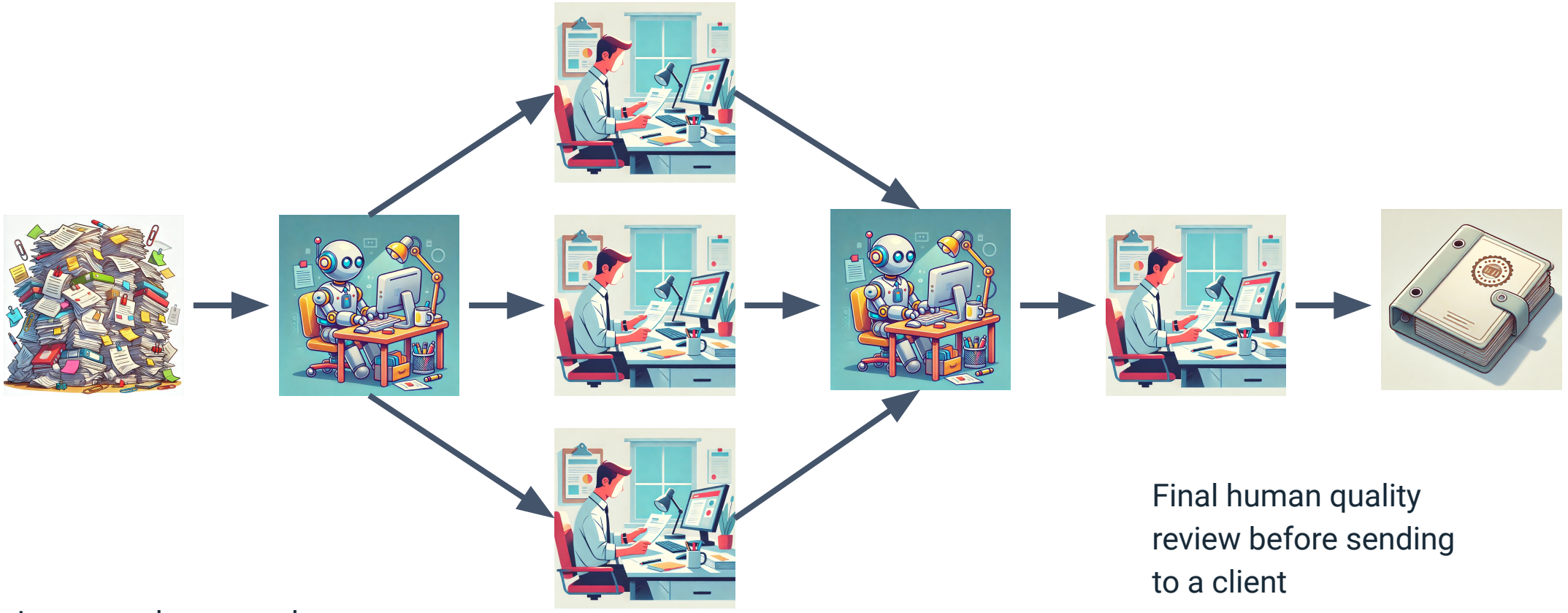


Highly trained analyst
doing end-to-end work



Finished product ready to
send to a client

The Solution: Assembly Lines



Inputs and prep work
from automated systems

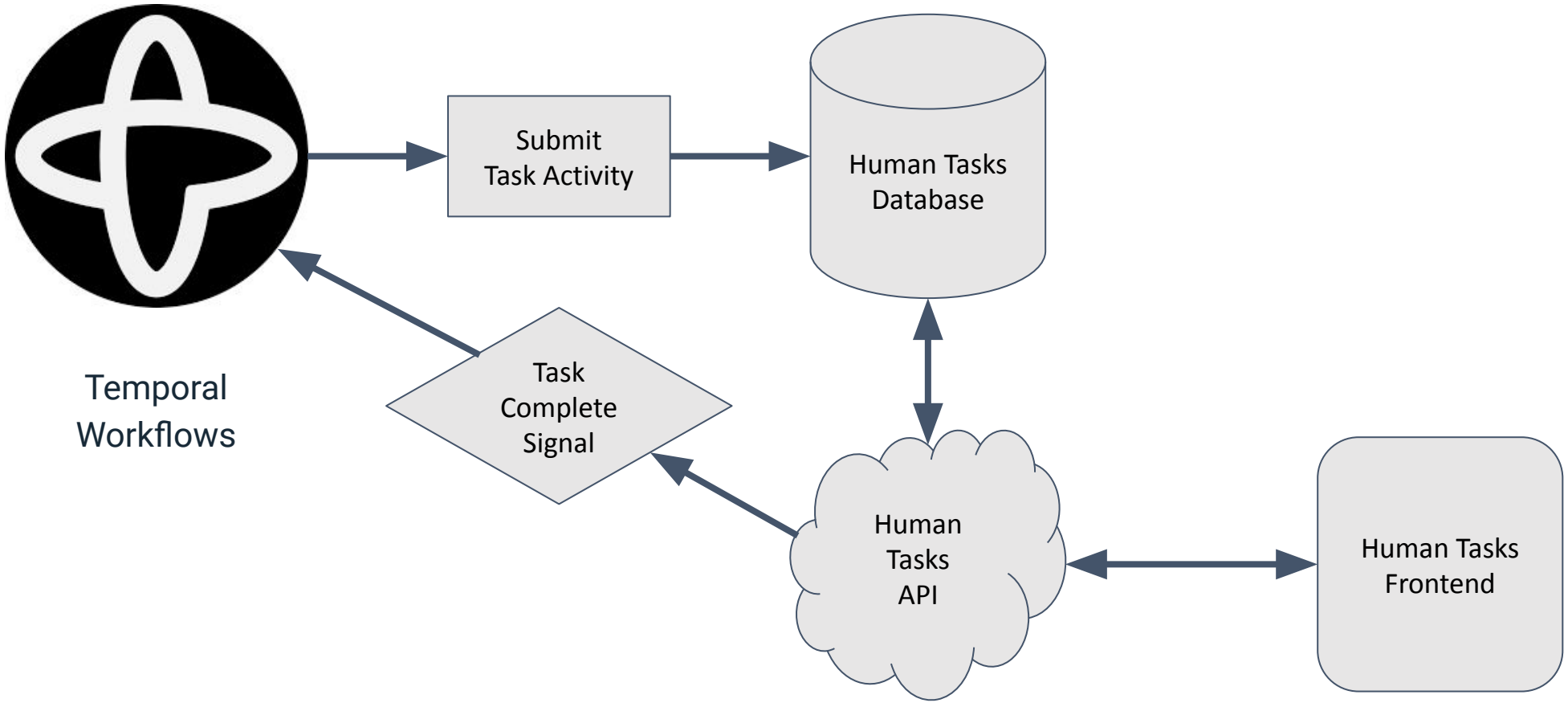
Mixed workflow of small
human tasks and
automated systems

Final human quality
review before sending
to a client

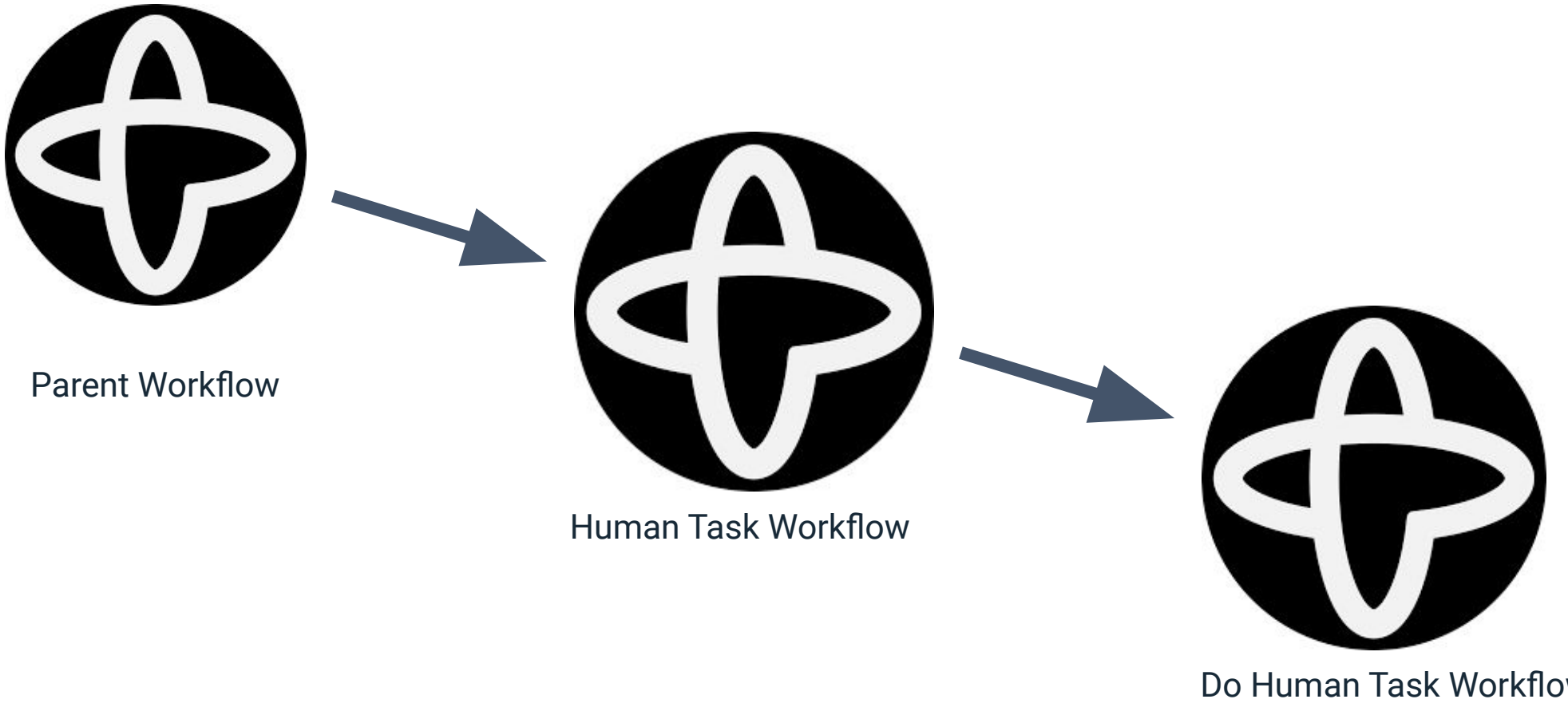
Solution Goals

- 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



Temporal Workflows

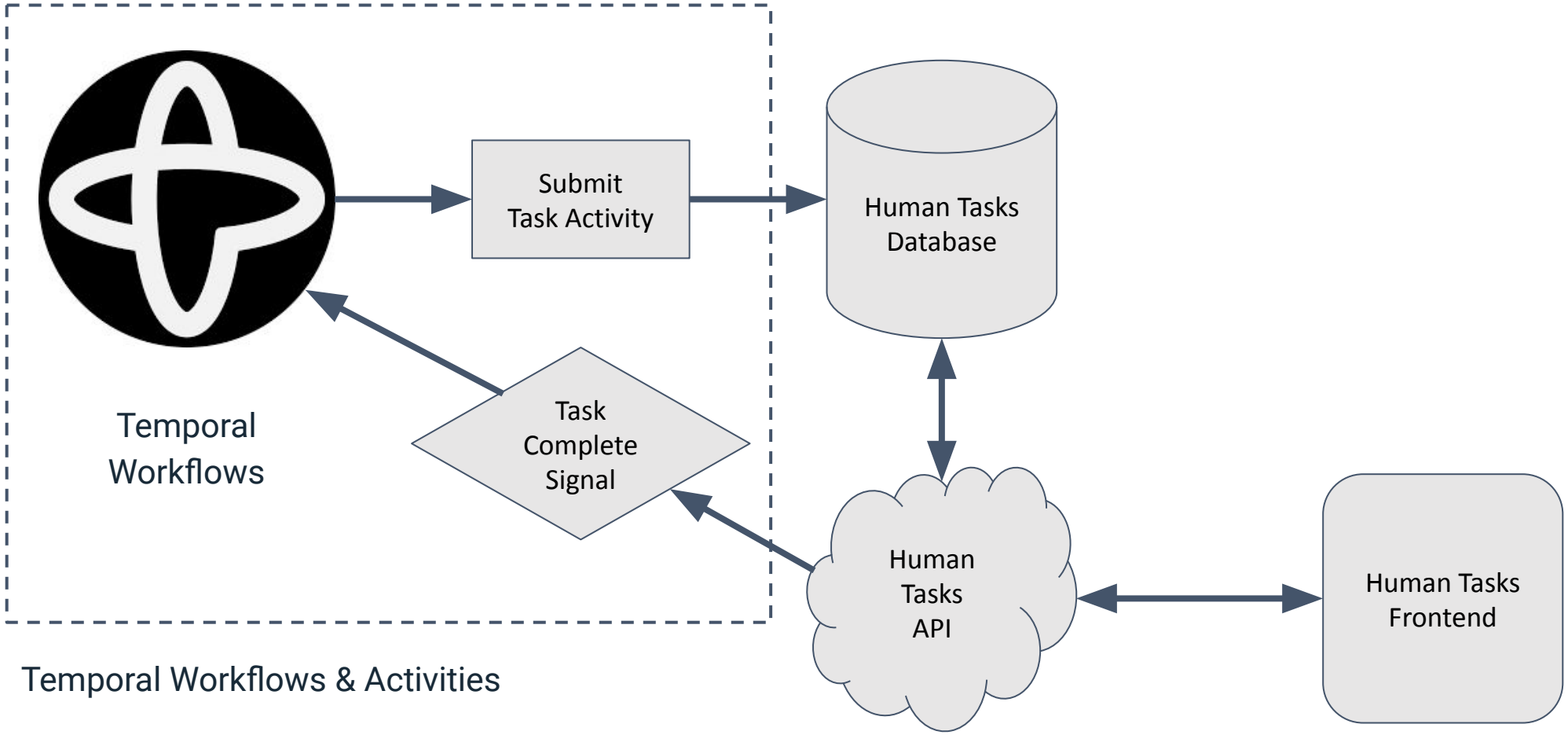


Sample Workflow

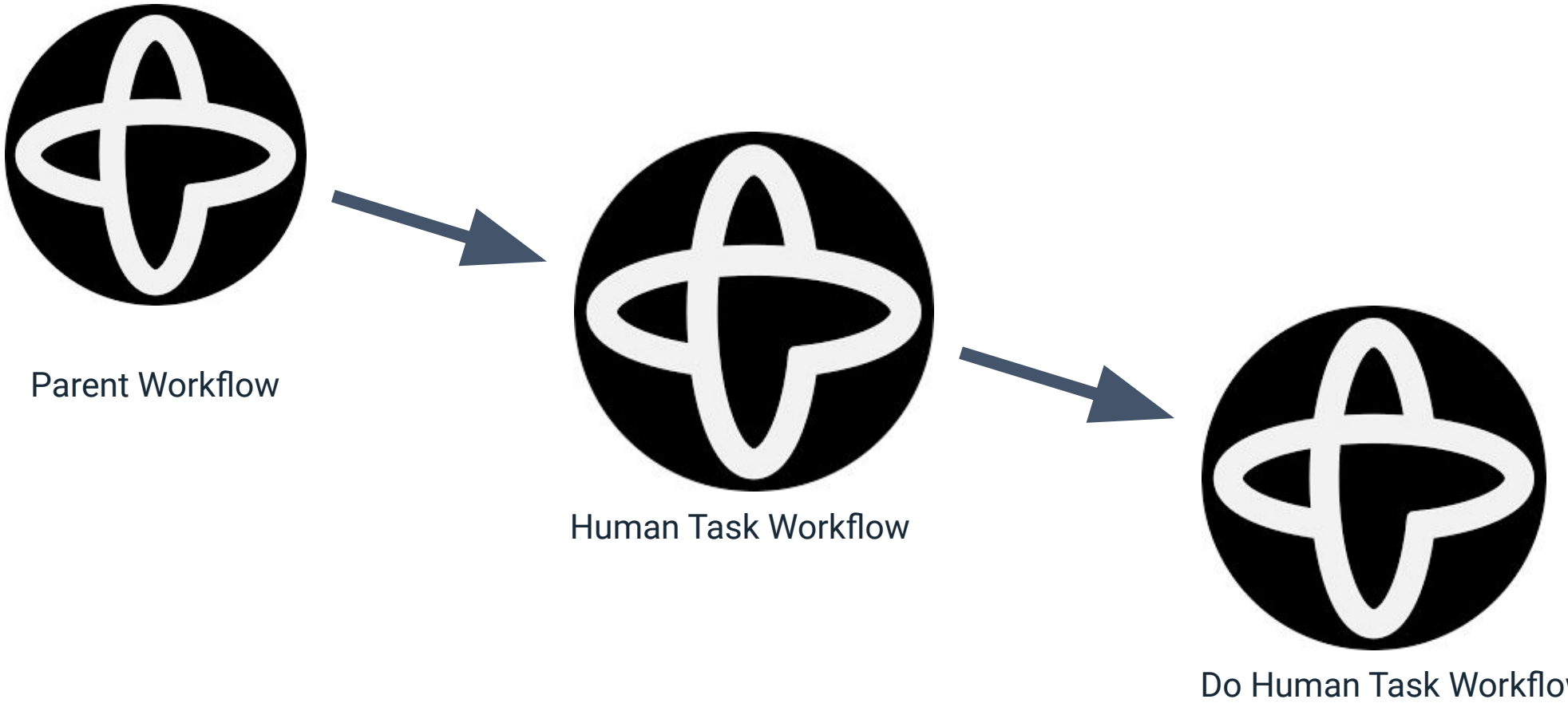
Add Digits Together

- A very pointless workflow that…
  - Accepts a string as input
  - Extracts all the digits from the input string
  - Asks a person to add the digits together
  - Returns the sum

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 addDigitsInStringTogether(
 input: string
): Promise<number> {
 const digits = await getDigitsFromString(input);

 const humanTaskResult = await addDigitsHumanTask({ digits });

 return doubleNumber(humanTaskResult.sum);
}
```

Task Wrapper Function

```
export async function addDigitsHumanTask(
 input: AddDigitsHumanTask["input"]
): Promise<AddDigitsHumanTask["output"]> {
 const humanTaskResult = await executeChild(humanTask<AddDigitsHumanTask>, {
   workflowId: `humanTask-${workflowInfo().workflowId}-add-digits-${uuid4()}`,
   workflowRunTimeout: "7d",
   args: ["add-digits", input],
 });

 return humanTaskResult;
}
```

Temporal Workflows

Human Task 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<Task extends HumanTask>(
 type: Task["type"],
 input: Task["input"]
): Promise<Task["output"]> {
 let result: HumanTaskCompletedSignalPayload | undefined;
 setHandler(humanTaskCompletedSignal, (paylaod) => { result = paylaod; });

 await signalWithStartHumanTask(type, input);
 await condition(() => result !== undefined);

 if (result!.success) return result!.output as Task["output"];
 else throw ApplicationFailure.fromError(result!.error);
}
```

Signal With Start Activity

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

 const output = await db.getTaskResult(workflowId);
 if (result) {
   client.workflow
     .getHandle(ctx.info.workflowExecution.workflowId)
     .signal(humanTaskCompletedSignal, { success: true, output });
   return;
 }
…
}
```

Signal With Start Activity

```
export async function signalWithStartHumanTask(args: TaskInput): Promise<void>{
 …
 await client.workflow.signalWithStart(doHumanTask, {
   workflowId,
   workflowRunTimeout: '7d',
   taskQueue: ctx.info.taskQueue,
   args: [args],
   signal: subscribeToHumanTaskCompletedSignal,
   signalArgs: [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 {
  // …
   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 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, (payload) => {
   result = payload
 })

 await signalWithStartHumanTask(args)

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

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

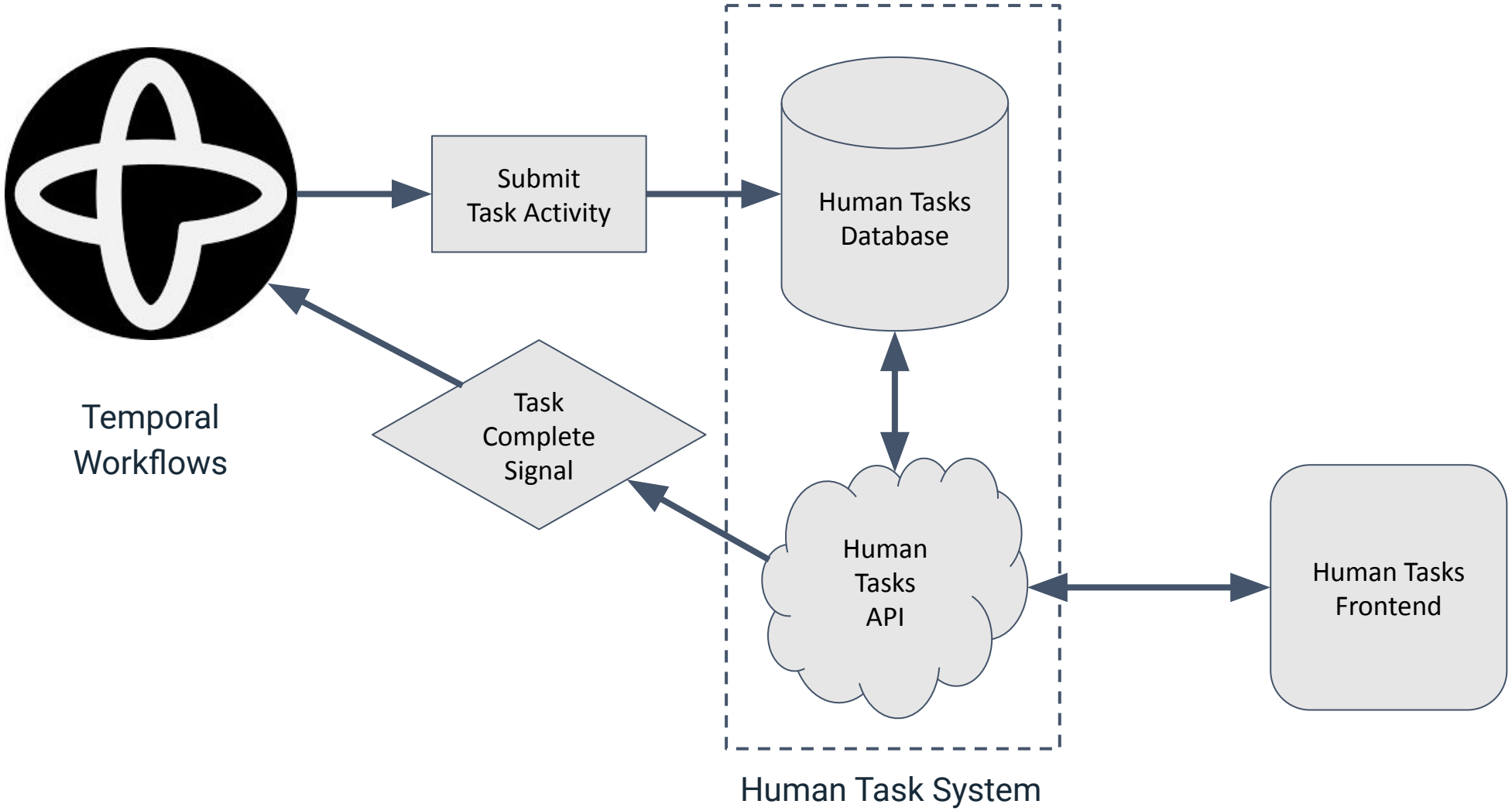
Parent Workflow

```
export async function addDigitsInStringTogether(
 input: string
): Promise<number> {
 const digits = await getDigitsFromString(input);

 const humanTaskResult = await addDigitsHumanTask({ digits });

 return doubleNumber(humanTaskResult.sum);
}
```

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

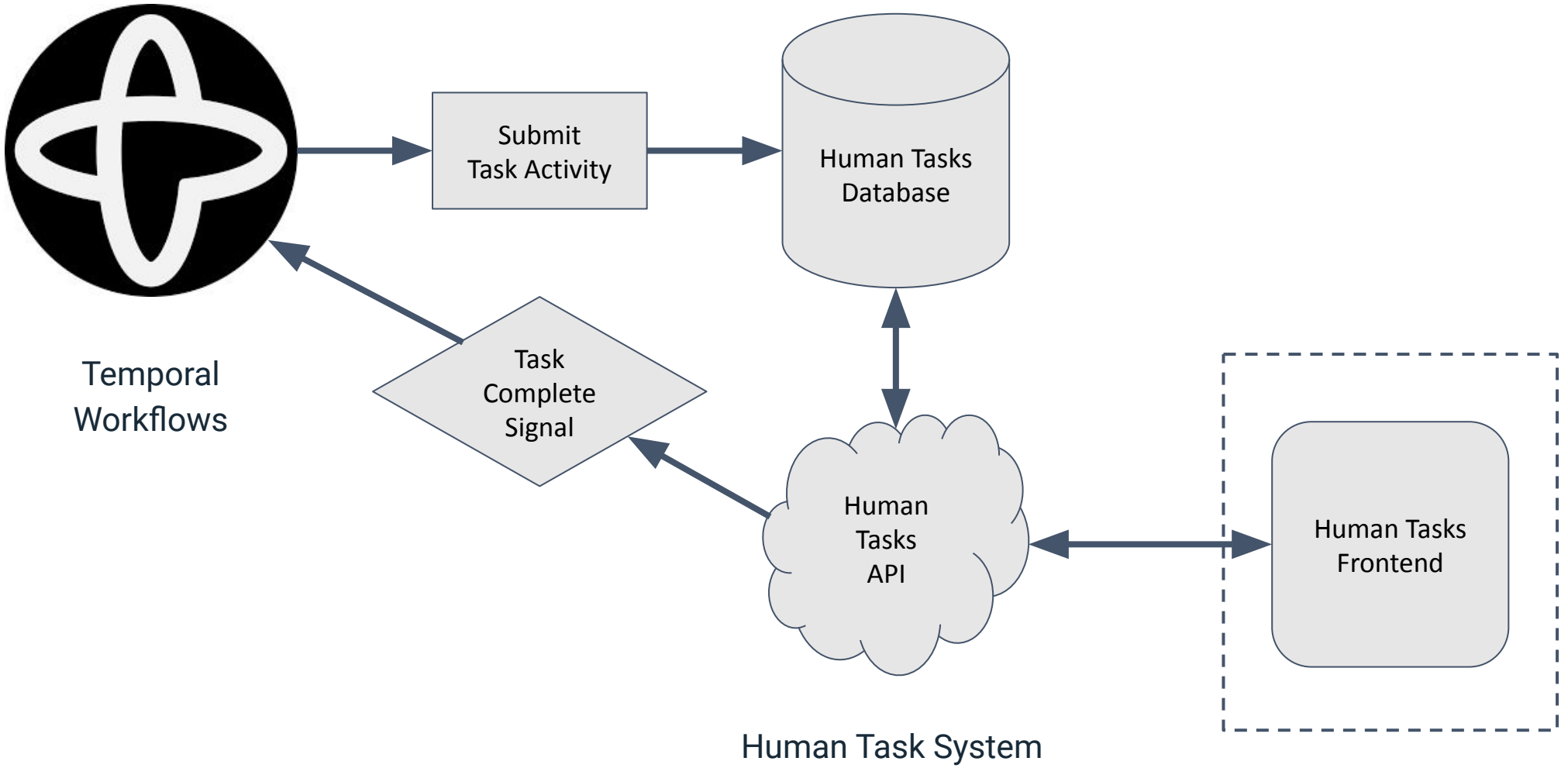
Heartbeat a Task

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

Complete a Task

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

High Level Design



Not even going to get into it...

- Each task will need a dedicated frontend
- Can be simple or complicated
- Need to heartbeat while work is ongoing

Results?

Results!

- Generating Citation Sheets
  - Documents parsed in hours instead of days
  - Reduced errors due to automation
  - Able to alter workflows in code instead of instructions

Future Projects - General

- Switch to updates for all task interactions
- Use data encoder / converter for all communications

Future Projects - Juristat

- More workflows!
- External / Anonymous assignees
- Task reviews
- Task training

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