WHAT IS DURABLE EXECUTION?

Temporal

WHAT IS DURABLE EXECUTION?

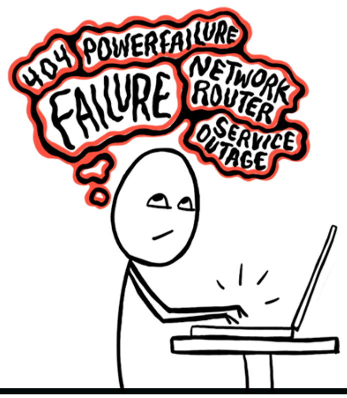
Developers like to write code.

Developers don't like when their code doesn't work.

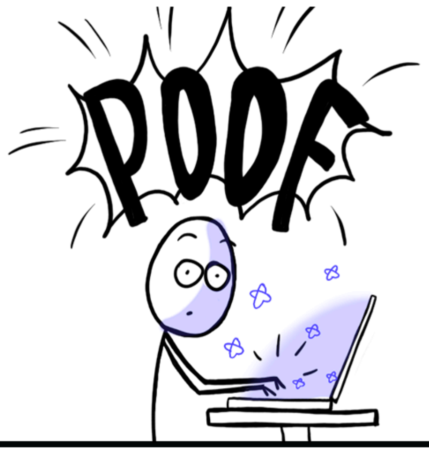
Sometimes, their code doesn't work because of their mistake.

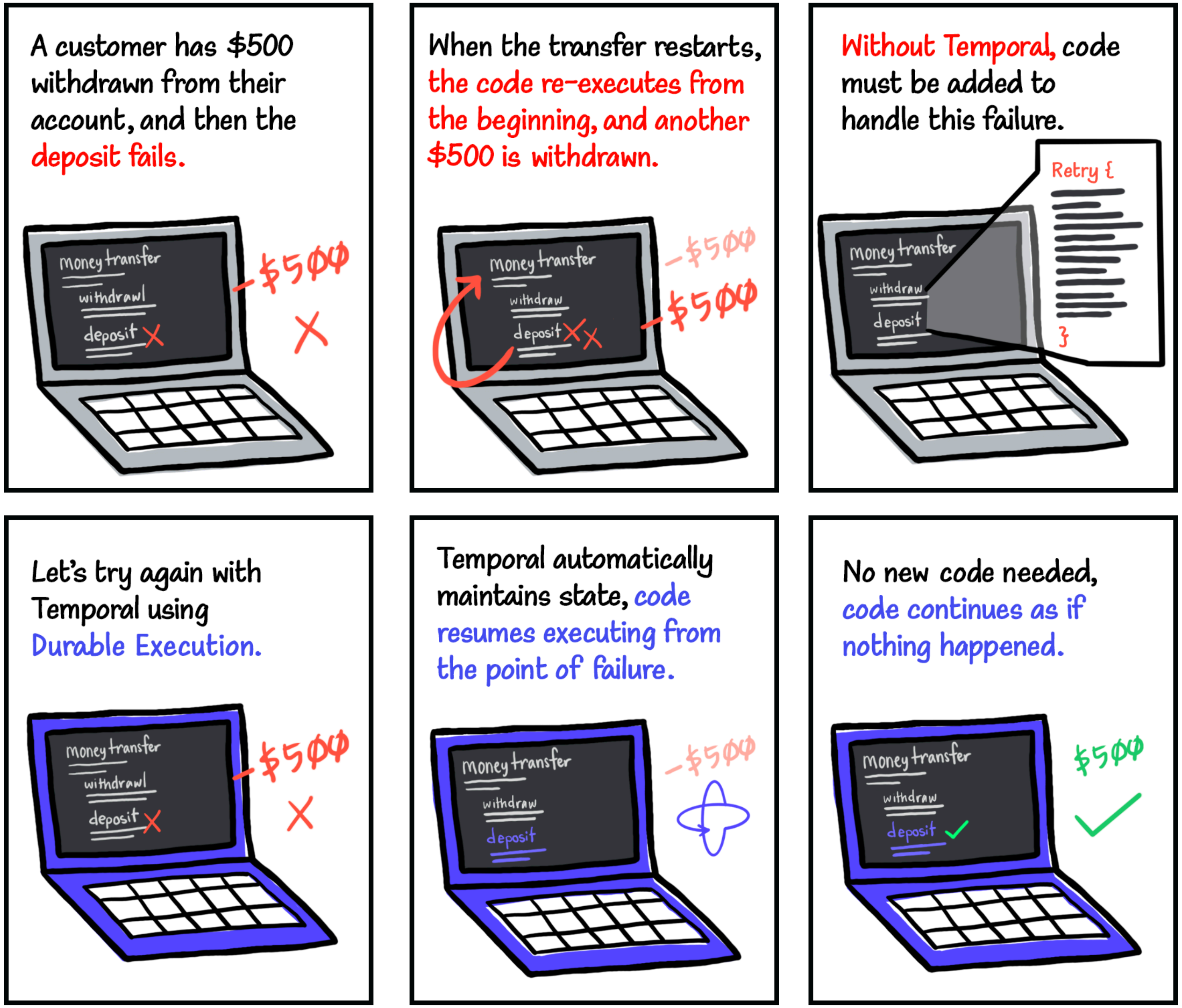
Other times it's not their fault, it's an external service or the network.

Developers spend a lot of time writing code to handle failures.



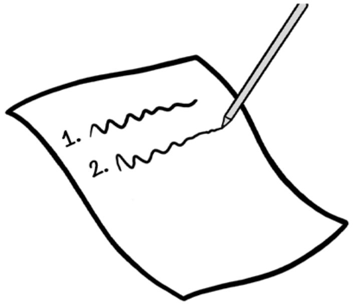
Durable Execution lets you code as if these failures don't exist.



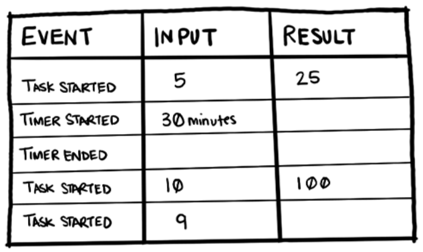

A customer has $500 withdrawn from their account, and then the deposit fails.
money transfer
withdrawl
deposit
-$5ØØ
When the transfer restarts, the code re-executes from the beginning, and another $500 is withdrawn.
money transfer
withdraw
deposit
-$5ØØ
-$5ØØ
Without Temporal, code must be added to handle this failure.
money transfer
withdraw
deposit
Retry {
}
Let's try again with Temporal using Durable Execution.
money transfer
withdrawl
deposit
-$5ØØ
Temporal automatically maintains state, code resumes executing from the point of failure.
money transfer
withdraw
deposit
-$5ØØ
No new code needed, code continues as if nothing happened.
money transfer
withdraw
deposit
$5ØØ

How Does Durable Execution Solve This Problem?

Temporal achieves Durable Execution by maintaining state in an Event History.

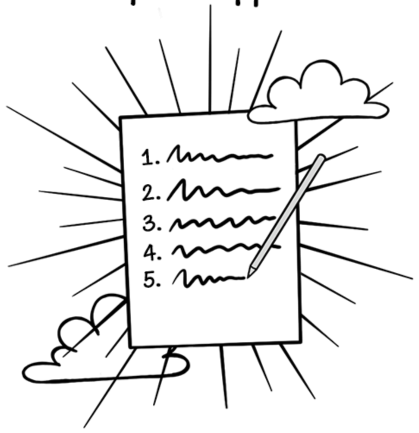


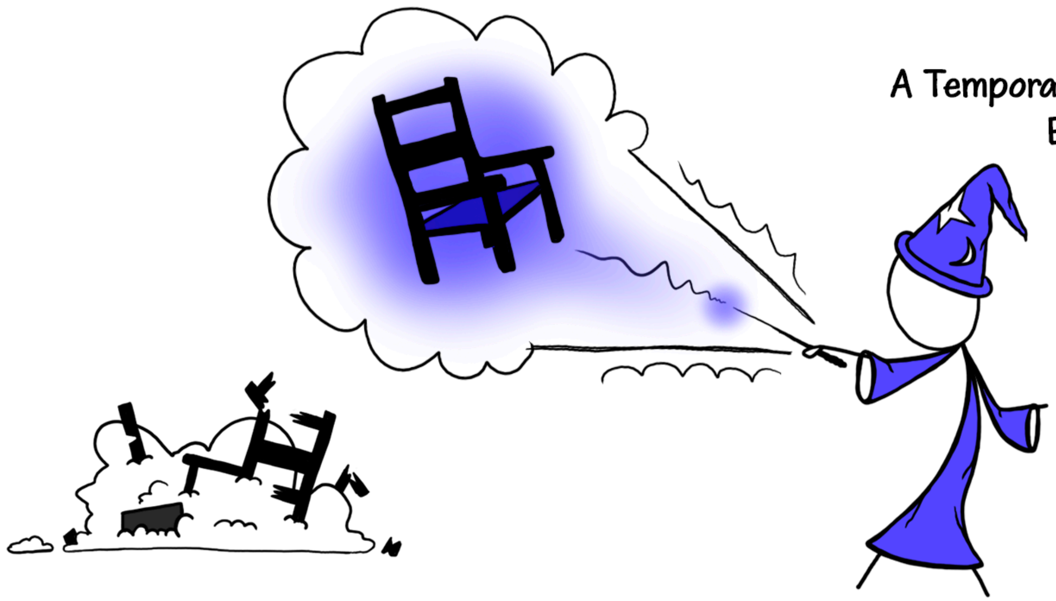
As each task completes, its inputs and results are recorded into the Event History.



EVENT	INPUT	RESULT
TASK STARTED	5	25
TIMER STARTED	30 minutes	
TIMER ENDED		
TASK STARTED	10	100
TASK STARTED	9	

The Event History is the source of truth for a Temporal application.





A Temporal application uses the Event History to automatically recover from failures.

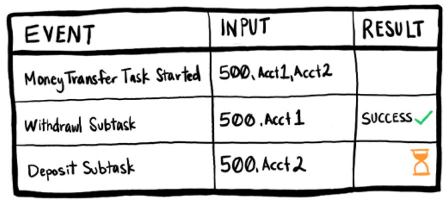
How Does the Event History Guarantee Durable Execution?

temporal.io

Say you have withdrawn $500 from the customer's account and now you're transferring to an external account.

money transfer
withdrawl ✓
deposit ⌛

These transactions have been recorded in the Temporal Event History.

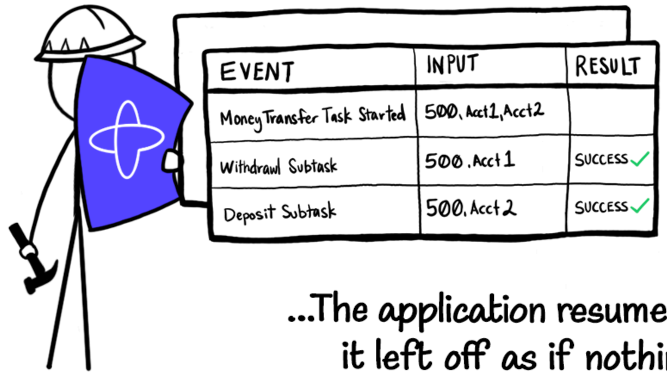


EVENT	INPUT	RESULT
Money Transfer Task Started	500, Acct1, Acct2	
Withdrawl Subtask	500, Acct 1	SUCCESS ✓
Deposit Subtask	500, Acct 2	⌛

During this execution, the datacenter experiences a power outage.

The power is restored and the application resumes execution.

A Temporal Application uses the Event History to reconstruct the state and recover from the failure. Previously completed tasks are not re-executed...

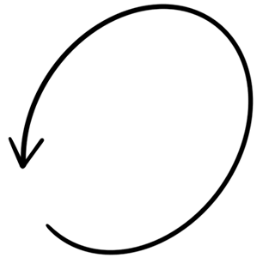


EVENT	INPUT	RESULT
Money Transfer Task Started	500, Acct1, Acct2	
Withdrawl Subtask	500, Acct 1	SUCCESS ✓
Deposit Subtask	500, Acct 2	SUCCESS ✓

...The application resumes from where it left off as if nothing happened.

What Happens if an External Service Goes Down?

Another way Temporal achieves Durable Execution is by retrying a failing task until it gets a successful result.



Say your code was in the middle of running, and a third-party service went down.

EVENT	INPUT	RESULT
Money Transfer Task Started	500, Acct1, Acct2	
Withdrawl Subtask	500, Acct 1	✓
Deposit Subtask	500, Acct 2	✗

A normal application would fail here, but Temporal detects the failure and retries the task until it gets a result.

EVENT	INPUT	RESULT
Money Transfer Task Started	500, Acct1, Acct2	
Withdrawl Subtask	500, Acct 1	✓
Deposit Subtask	500, Acct 2	✢

No need to retry the entire transaction.

Eventually the third-party application comes back online, and the service is able to complete the request.

EVENT	INPUT	RESULT
Money Transfer Task Started	500, Acct1, Acct2	
Withdrawl Subtask	500, Acct 1	✓
Deposit Subtask	500, Acct 2	✓

The execution continued, and the user was unaware there was an outage.

So was the programmer.

Temporal handles this for you, saving the programmer time and making them happy.

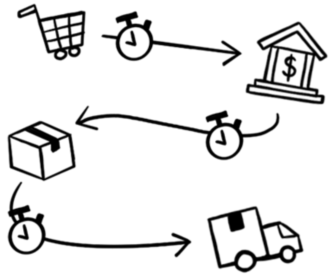
Here is an example of when you would benefit from Durable Execution. Consider the lifecycle of an e-commerce order.

USE CASE: ACCESSING SERVICES VIA THE NETWORK.

Your process relies on services available via the network.

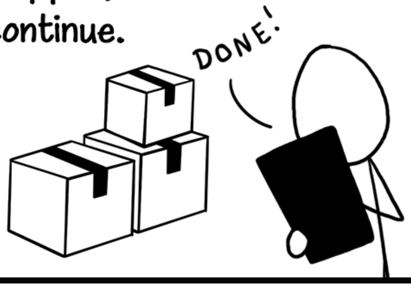
USE CASE: LONG RUNNING TASKS.

From order to delivery to possible returns, this process could take days or even weeks.



USE CASE: HUMAN-IN-THE-LOOP

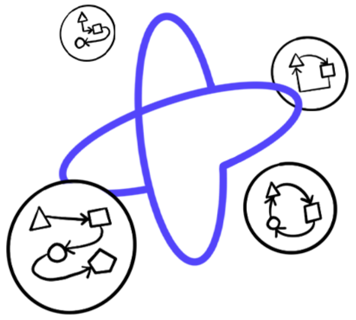
Certain parts of the process are held up until a human confirms that it is done, such as packing the item. Once the item is shipped, the workflow can continue.



USE CASE: RECOVERING FROM FAILURE.

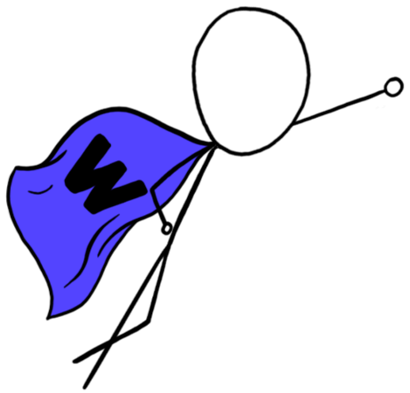
If a step fails, we must undo the ones before it to keep system state consistent.

Temporal provides mechanisms that make implementing these use cases easier.

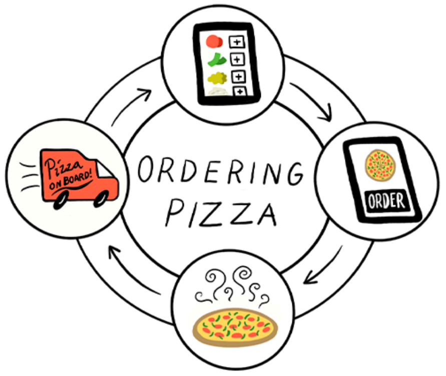


How Does Temporal Provide Durable Execution?

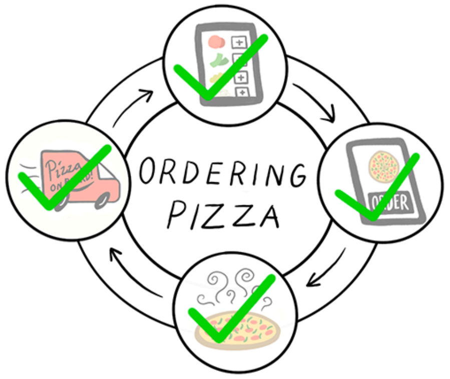
Temporal provides Durable Execution through Workflows.



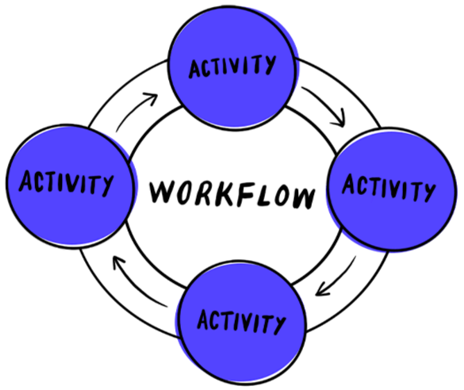
Workflows are a sequence of steps taken to perform a task.



Workflows are guaranteed to run to completion and must be deterministic.



Workflows can be composed of smaller actions called Activities.



Activities are operations that are prone to failure (such as calling APIs, writing to databases, etc.). They need not be deterministic.

By default, Activities are retried on failure.

GET STARTED AT LEARN.TEMPORAL.IO WHERE YOU'LL FIND COURSES, PROJECT-BASED TUTORIALS AND MORE.