The thunk trap

Why your Nix loop crashes after it finishes

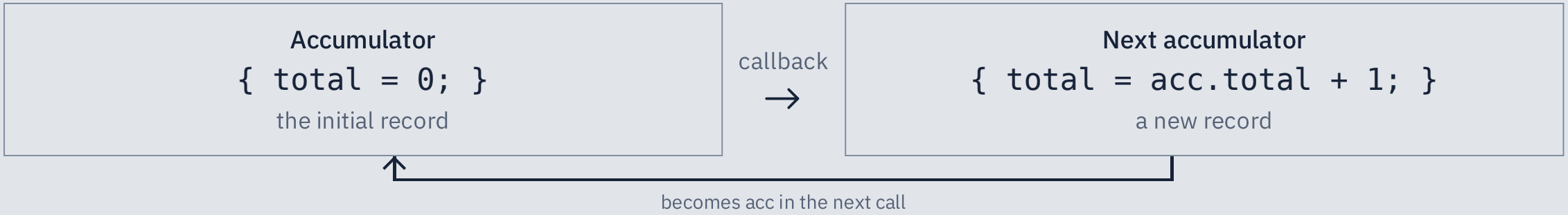
Mika Bohinen - NixCon 2026

The fold passes each returned record to the next call.

The callback, folded over ticks: 100000 inputs

```
acc: _: { total = acc.total + 1; }
```

One call: the callback builds the next record



The fold produces a record, but reading its total overflows.

result is the record returned by a fold that adds one to a total 100000 times.

Which fields are in the result?

```
builtins.attrNames result
```

```
["total"]
```

Recorded result

What is the final total?

```
result.total
```

```
error: stack overflow (possible
infinite recursion)
```

Recorded result

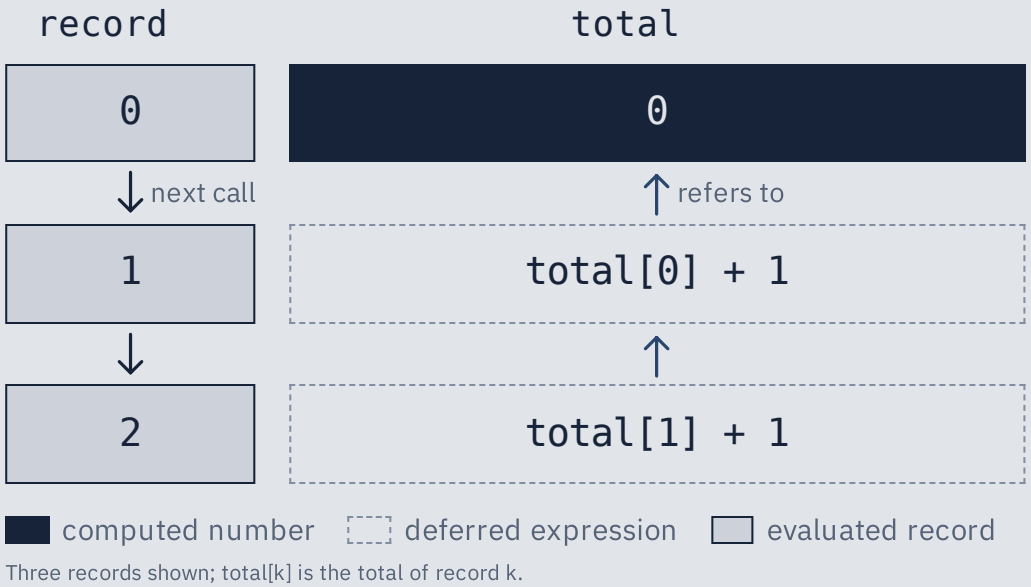
Having the record does not mean its total has been computed.

foldl' evaluates each record, but its total can remain unevaluated.

One record per input; ticks has 100000 items

```
result = builtins.foldl'
  (acc: _: { total = acc.total + 1; })
  { total = 0; }
  ticks;
```

foldl' evaluates each record; its total stays deferred



Three records shown; total[k] is the total of record k.

```
builtins.attrNames result                                    ["total"]
```

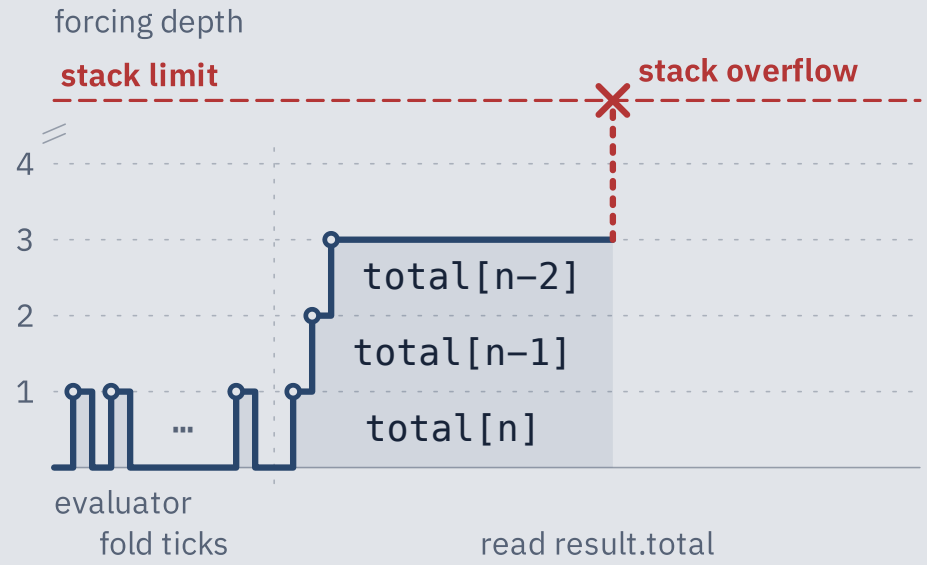
Recorded result

Reading the final total follows a chain of deferred additions.

One record per input; ticks has 100000 items

```
result = builtins.foldl'
  (acc: _: { total = acc.total + 1; })
  { total = 0; }
  ticks;
```

Reading the final total climbs through every deferred addition



Illustrative trace for n steps; the recorded run overflows at n = 100000.

`result.total` → `error: stack overflow (possible infinite recursion)`

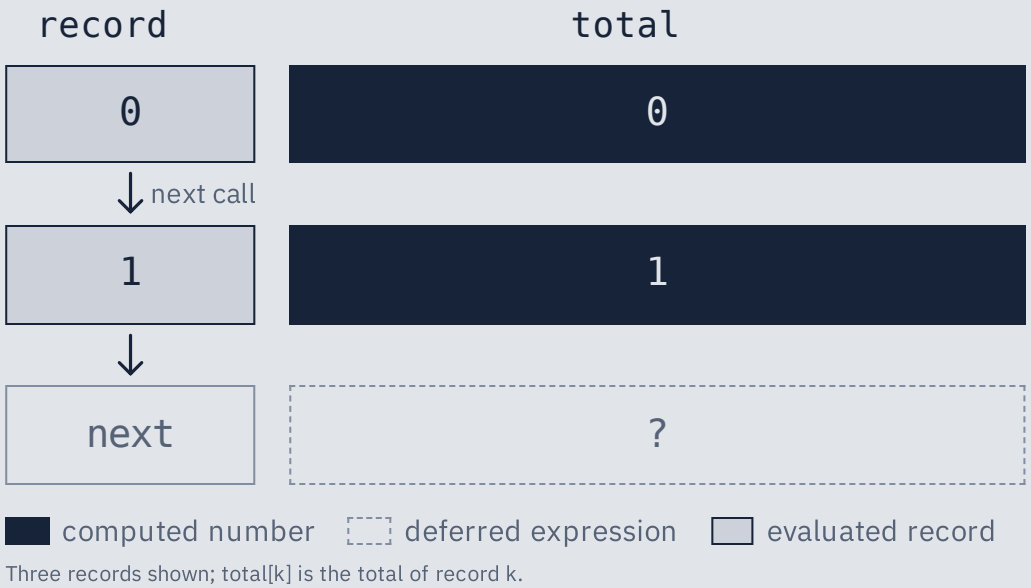
Recorded result

Compute the total before returning the next record.

countStep: replacement callback for foldl'

```
acc: _:
  let total = acc.total + 1;
  in builtins.seq total { inherit total; }
```

First call: compute 1 before returning the record



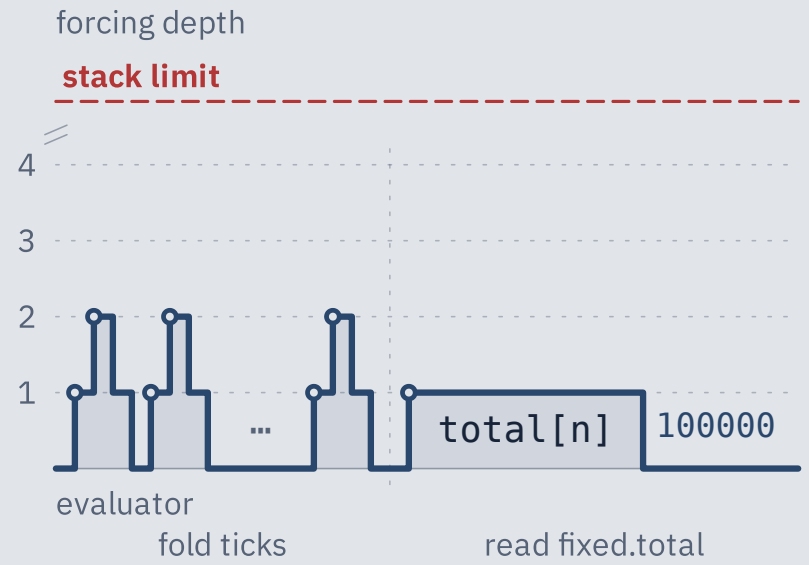
Three records shown; total[k] is the total of record k.

The final total now succeeds with the same input and limits.

fixed = builtins.foldl' countStep { total = 0; } ticks

```
acc: _:
  let total = acc.total + 1;
  in builtins.seq total { inherit total; }
```

Each step computes its total before the next begins



Illustrative trace for n steps; the recorded run returns 100000.

`fixed.total`	`100000`

Recorded result

deepSeq also fixes this counter by evaluating the record's fields.

Another replacement callback

```
acc: _:
  let next = { total = acc.total + 1; };
  in builtins.deepSeq next next
```

Demand	Evaluates	Still deferred
`seq next next`	the record	total
`seq next.total next`	total	nothing
`deepSeq next next`	every field, including total	nothing

deepRecordFold: total from the deepSeq callback

`deepRecordFold` `100000`

Recorded result

seq next.total computes the number without evaluating metadata.

metadata throws if it is demanded

```
next = {
  total = 1 + 1;
  metadata = throw "not needed for counting";
};
```

```
(builtins.seq next.total next).total                    2
```

Recorded result

```
(builtins.deepSeq next next).total                      error: not needed for counting
```

Recorded result

Compute the running total before returning the next record.

countStep

```
acc: _:
  let total = acc.total + 1;
  in builtins.seq total { inherit total; }
```

nixcon2026.bohinen.no

Examples - captured evidence - reading - PDFs

sternenseemann: trampolining Nix (2022)