1 `test_sum_to_n` — MIR Walkthrough

Purpose: TODO: Describe why this walkthrough exists

1.1 Source Context

```
    let sucess = sum_to_n(n) == golden;
    assert!(sucess);
}
```

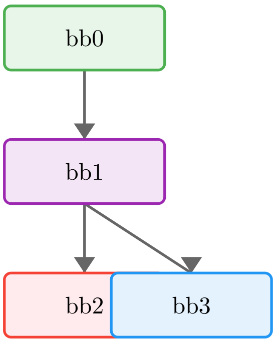
1.2 Function Overview

- **Function:** `test_sum_to_n`
- **Basic blocks:** 4
- **Return type:** `()`
- **Notable properties:**
  - Contains panic path
  - Has conditional branches

1.3 Locals

Local	Type	Notes
0	`()`	Return place
1	`bool`	
2	`usize`	
3	`usize`	
4	`usize`	
5	`!`	

1.4 Control-Flow Overview



1.5 Basic Blocks

1.5.1 bb0 — entry

Entry point of the function.

MIR	Annotation
`\_3 = 10`	Load constant
`→ \_2 = sum\_to\_n(move \_3) → bb1`	Call sum_to_n

1.5.2 bb1 — branch point

MIR	Annotation

`\_4 = 55`	Load constant
`\_1 = move \_2 == move \_4`	Equal operation
`→ switch(\_1) \[0→bb2; else→bb3\]`	Branch on _1

1.5.3 bb2 — panic path

Panic/diverging path.

MIR	Annotation
`→ \_5 = panic(\[16 bytes\])`	Call panic

1.5.4 bb3 — return / success

Normal return path.

MIR	Annotation
`→ return`	Return from function

1.6 Key Observations

TODO: Add bullet points summarizing what this MIR teaches

-
-

1.7 Takeaways

TODO: One or two sentences to generalize this example

2 main — MIR Walkthrough

Purpose: TODO: Describe why this walkthrough exists

2.1 Source Context

```
fn main() {
  test_sum_to_n();
  return ();
}
```

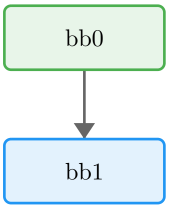
2.2 Function Overview

- **Function:** `main`
- **Basic blocks:** 2
- **Return type:** `()`

2.3 Locals

Local	Type	Notes
`0`	`()`	Return place
`1`	`()`	

2.4 Control-Flow Overview



2.5 Basic Blocks

2.5.1 bb0 — entry

Entry point of the function.

MIR	Annotation
`→ \_1 = test\_sum\_to\_n() → bb1`	Call test_sum_to_n

2.5.2 bb1 — return / success

Normal return path.

MIR	Annotation
`→ return`	Return from function

2.6 Key Observations

TODO: Add bullet points summarizing what this MIR teaches

-
-

2.7 Takeaways

TODO: One or two sentences to generalize this example

3 sum_to_n — MIR Walkthrough

Purpose: TODO: Describe why this walkthrough exists

3.1 Source Context

```
fn sum_to_n(n:usize) -> usize {
    let mut sum = 0;
    let mut counter = n;

    while counter > 0 {
      sum += counter;
      counter = counter - 1;
    }
    return sum;
}
```

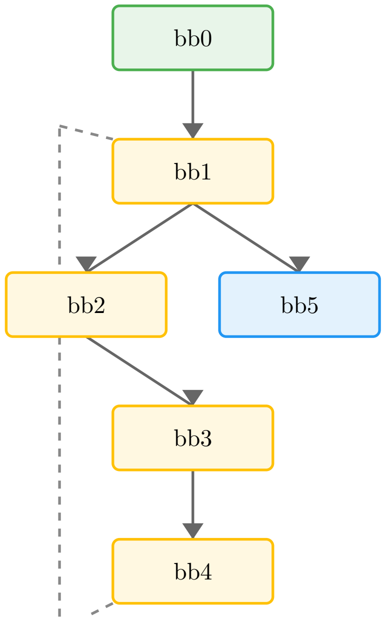
3.2 Function Overview

- **Function:** `sum_to_n`
- **Basic blocks:** 6
- **Return type:** `usize`
- **Notable properties:**
  - Contains panic path
  - Uses checked arithmetic
  - Contains assertions
  - Has conditional branches

3.3 Locals

Local	Type	Notes
0	`usize`	Return place
1	`usize`	
2	`usize`	
3	`usize`	
4	`bool`	
5	`usize`	
6	`usize`	
7	`(usize, bool)`	
8	`usize`	
9	`(usize, bool)`	

3.4 Control-Flow Overview



3.5 Basic Blocks

3.5.1 bb0 — entry

Entry point of the function.

MIR	Annotation
`\_2 = 0`	Load constant
`\_3 = \_1`	Copy value
`→ goto bb1`	Jump to bb1

3.5.2 bb1 — loop

MIR	Annotation
`\_5 = \_3`	Copy value
`\_4 = move \_5 \> 0`	Greater than operation
`→ switch(move \_4) \[0→bb5; else→bb2\]`	Branch on move _4

3.5.3 bb2 — loop

MIR	Annotation
`\_6 = \_3`	Copy value
`\_7 = checked(\_2 + \_6)`	Checked Add (may panic)
`→ assert(move \_7.1 == false) → bb3`	Panic if move _7.1 is true

3.5.4 bb3 — loop

MIR	Annotation
`\_2 = move \_7.0`	Move value
`\_8 = \_3`	Copy value
`\_9 = checked(\_8 - 1)`	Checked Subtract (may panic)
`→ assert(move \_9.1 == false) → bb4`	Panic if move _9.1 is true

3.5.5 bb4 — loop

MIR	Annotation
`\_3 = move \_9.0`	Move value
`→ goto bb1`	Jump to bb1

3.5.6 bb5 — return / success

Normal return path.

MIR	Annotation
`\_0 = \_2`	Copy value
`→ return`	Return from function

3.6 Key Observations

TODO: Add bullet points summarizing what this MIR teaches

-
-

3.7 Takeaways

TODO: One or two sentences to generalize this example