1 `main` — MIR Walkthrough

Purpose: TODO: Describe why this walkthrough exists

1.1 Source Context

```
fn main() {
    let ans = fibonacci(5);

    assert!(ans == 5);
}
```

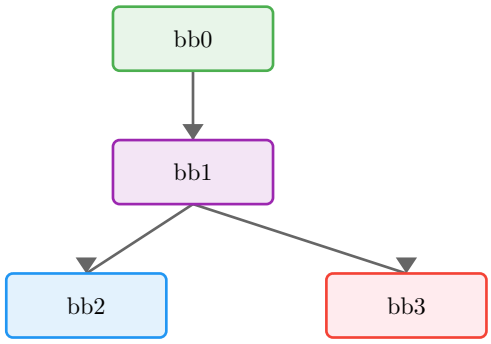
1.2 Function Overview

- **Function:** `main`
- **Basic blocks:** 4
- **Return type:** `() (0 bytes, align 1)`
- **Notable properties:**
  - Contains panic path
  - Has conditional branches

1.3 Locals

Local	Type	Notes
0	`() (0 bytes, align 1)`	Return place
1	`Uint(U32)`	
2	`()`	

1.4 Control-Flow Overview



1.5 Basic Blocks

1.5.1 bb0 — entry

Entry point of the function.

MIR	Annotation
`→ _1 = fibonacci(5) → bb1`	Call fibonacci

1.5.2 bb1 — branch point

MIR	Annotation
`→ switch(_1) [5→bb2; else→bb3]`	Branch on _1

1.5.3 bb2 — return / success

Normal return path.

MIR	Annotation
`→ return`	Return from function

1.5.4 bb3 — panic path

Panic/diverging path.

MIR	Annotation
`→ _2 = panic([16 bytes])`	Call panic

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 `fibonacci` — MIR Walkthrough

Purpose: TODO: Describe why this walkthrough exists

2.1 Source Context

```
fn fibonacci(n:u32) -> u32 {
    match n {
        0 => 0,
        1 => 1,
        _ => fibonacci(n - 2) + fibonacci(n - 1),
    }
}
```

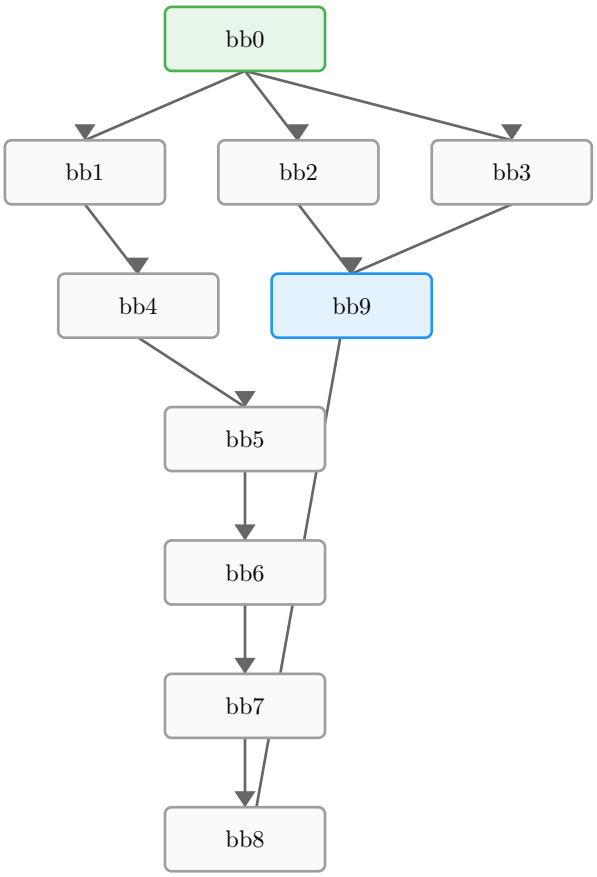
2.2 Function Overview

- **Function:** `fibonacci`
- **Basic blocks:** 10
- **Return type:** `Uint(U32)`
- **Notable properties:**
  - Contains panic path
  - Uses checked arithmetic
  - Recursive
  - Contains assertions
  - Has conditional branches

2.3 Locals

Local	Type	Notes
0	`Uint(U32)`	Return place
1	`Uint(U32)`	
2	`Uint(U32)`	
3	`Uint(U32)`	
4	`(u32, bool) (8 bytes, align 4)`	
5	`Uint(U32)`	
6	`Uint(U32)`	
7	`(u32, bool) (8 bytes, align 4)`	
8	`(u32, bool) (8 bytes, align 4)`	

2.4 Control-Flow Overview



2.5 Basic Blocks

2.5.1 bb0 — entry

Entry point of the function.

MIR	Annotation
`→ switch(_1) [0→bb3, 1→bb2; else→bb1]`	Branch on _1

2.5.2 bb1

MIR	Annotation
`_4 = checked(_1 - 2)`	Checked Subtract (may panic)
`→ assert(move _4.1 == false) → bb4`	Panic if move _4.1 is true

2.5.3 bb2

MIR	Annotation
`_0 = 1`	Load constant
`→ goto bb9`	Jump to bb9

2.5.4 bb3

MIR	Annotation
`_0 = 0`	Load constant
`→ goto bb9`	Jump to bb9

2.5.5 bb4

MIR	Annotation
`_3 = move _4.0`	Move value
`→ _2 = fibonacci(move _3) → bb5`	Recursive call to fibonacci

2.5.6 bb5

MIR	Annotation
`_7 = checked(_1 - 1)`	Checked Subtract (may panic)
`→ assert(move _7.1 == false) → bb6`	Panic if move _7.1 is true

2.5.7 bb6

MIR	Annotation
`_6 = move _7.0`	Move value
`→ _5 = fibonacci(move _6) → bb7`	Recursive call to fibonacci

2.5.8 bb7

MIR	Annotation
`_8 = checked(_2 + _5)`	Checked Add (may panic)
`→ assert(move _8.1 == false) → bb8`	Panic if move _8.1 is true

2.5.9 bb8

MIR	Annotation
`_0 = move _8.0`	Move value
`→ goto bb9`	Jump to bb9

2.5.10 bb9 — 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