



1000X

WORLD TOUR

13 CITIES | 6 DAYS



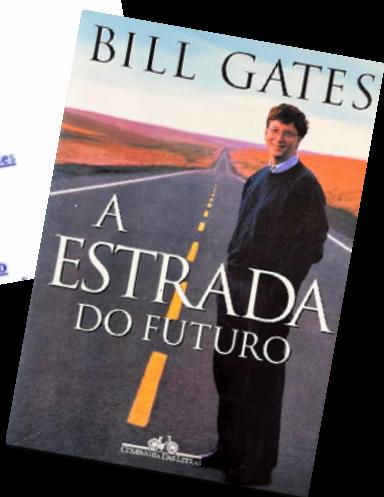


1000X

WORLD TOUR

13 CITIES | 6 DAYS

1000X



1000X

OLAP

Online Analytical Processing

1000X

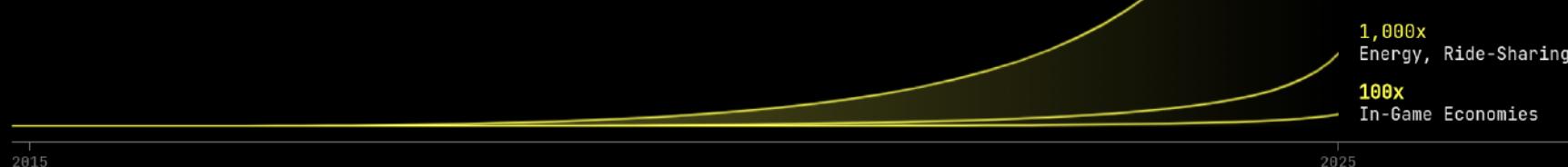
“A história não se repete, mas rima.”

Mark Twain

1000X

The World is Becoming More Transactional

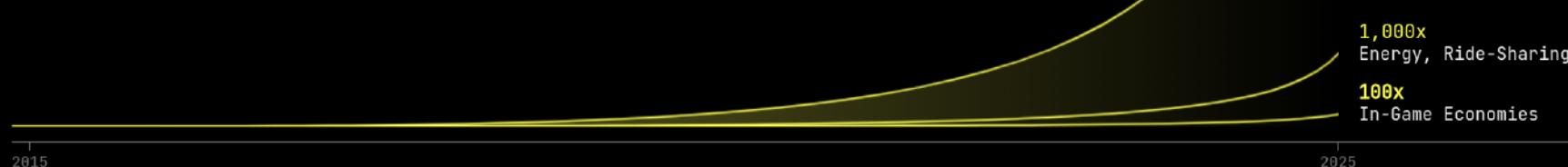
In the past 10 years, transaction volumes surged 100x–1,000x across cloud, energy, and gaming — and up to 10,000x in real-time payments. Mainframe or not, legacy systems are at a breaking point.



1000X

The World is Becoming More Transactional

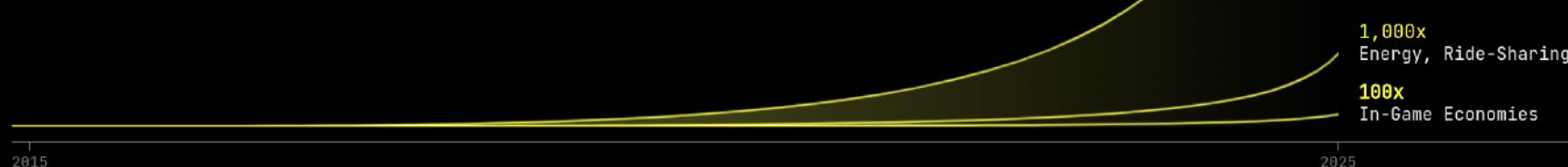
In the past 10 years, transaction volumes surged 100x–1,000x across cloud, energy, and gaming — and up to 10,000x in real-time payments. Mainframe or not, legacy systems are at a breaking point.



1000X

The World is Becoming More Transactional

In the past 10 years, transaction volumes surged 100x–1,000x across cloud, energy, and gaming — and up to 10,000x in real-time payments. Mainframe or not, legacy systems are at a breaking point.



1000X

Transações estão em toda parte!



1000X

Transações estão em toda parte!



1000x

OLTP

Online Transaction Processing

1000X

THE FINANCIAL TRANSACTIONS DATABASE ^{1000X} FASTER

To power the next 30 years of Online Transaction Processing.

Install TigerBeetle

Read the Docs



1000X

THE FINANCIAL TRANSACTIONS DATABASE ^{1000X} FASTER

To power the next 30 years of Online Transaction Processing.

Install TigerBeetle

Read the Docs



1000X

A filosofia por trás do processo de desenvolvimento do TigerBeetle

Rafael Batiati

Senior Engineer @ TigerBeetle

x.com/rbatati

github.com/batiati

1000x

10^1 = Systems Programming

1000X

10^1 = Systems Programming



Systems programming, or system programming, is the activity of programming^[1] computer system software. The primary distinguishing characteristic of systems programming when compared to application programming is that application programming aims to produce software which provides services to the user directly (e.g. word processor), whereas systems programming aims to produce software and software platforms which provide services to other software, are performance constrained, or both (e.g. operating systems, computational science applications, game engines, industrial automation, and software as a service applications).^[1]

1000X

10^1 = Systems Programming

*“A way of modeling
software development”*



Andrew Kelley
Criador da linguagem
de programação ZIG

<https://youtu.be/Qncdi-Fg0-I>



1000X

10^1 = Systems Programming

*“A way of modeling
software development”*



Andrew Kelley
Criador da linguagem
de programação ZIG

<https://youtu.be/Qncdi-Fg0-I>



1000X

10^1 = Systems Programming

*“A way of modeling
software development”*



Andrew Kelley
Criador da linguagem
de programação ZIG

<https://youtu.be/Qncdi-Fg0-I>



1000X

10^1 = Systems Programming

*“The purpose of abstracting is not to be vague,
but to create a new semantic level in which
one can be absolutely precise.”*

Edsger W. Dijkstra

1000X

10^1 = Systems Programming

*“The purpose of abstracting is not to be vague,
but to create a new semantic level in which
one can be absolutely precise.”*

Edsger W. Dijkstra

1000X

10^1 = Systems Programming

The Power of Zero Dependencies

1000X

10^1 = Systems Programming

The Power of Zero Dependencies

1000x

$10^2 = \text{Esforço} \ vs \ \text{Resultado}$

1000x

$10^2 = \text{Esforço} \ vs \ \text{Resultado}$

1000X

$10^2 = \text{Esforço} \text{ vs Resultado}$

Tempo = Design + Desenvolvimento + Testes + Incidentes

1000X

$10^2 = \text{Esforço} \text{ vs Resultado}$

Tempo = Design + Desenvolvimento + Testes + Incidentes

1000X

$10^2 = \text{Esforço} \text{ vs Resultado}$

Tempo = Design + Desenvolvimento + Testes + Incidentes

1000X

$10^2 = \text{Esforço} \text{ vs Resultado}$

Tempo = Design + Desenvolvimento + Testes + Incidentes

“Go slow to go fast”



1000X

$10^2 = \text{Esforço} \text{ vs Resultado}$

Tempo = Design + Desenvolvimento + Testes + Incidentes

1000X

$10^2 = \text{Esforço} \text{ vs Resultado}$

Tempo = Design + Desenvolvimento + Testes + Incidentes

1000X

$10^2 = \text{Esforço} \text{ vs Resultado}$

Tempo = Design + Desenvolvimento + Testes + Incidentes

1000X

$10^2 = \text{Esforço} \text{ vs Resultado}$

Tempo = Design + Desenvolvimento + Testes + Incidentes

1000X

$10^2 = \text{Esforço} \text{ vs Resultado}$

Tempo = Design + Desenvolvimento + Testes + Incidentes

1000X

$10^2 = \text{Esforço} \text{ vs Resultado}$

Tempo = Design + Desenvolvimento + Testes + Incidentes

TIGER STYLE



100

$10^2 =$

Tempo =

```
1 fn execute_expire_pending_transfers(self: *StateMachine, timestamp: u64) usize {
2     assert(timestamp > self.commit_timestamp);
3     assert(self.scan_lookup_results.items.len > 0);
4
5     // ...
6
7     assert(result_count <= result_max);
8     assert(self.scan_lookup_buffer_index > 0);
9     assert(self.scan_lookup_buffer_index == result_count * @sizeOf(Transfer));
10
11    for (transfers_pending, 0..) |*p, index| {
12        assert(p.flags.pending);
13        assert(p.timeout > 0);
14
15        const event_timestamp = timestamp - transfers_pending.len + index + 1;
16        assert(TimestampRange.valid(event_timestamp));
17        assert(self.commit_timestamp < event_timestamp);
18
19        const expires_at = p.timestamp + p.timeout_ns();
20        assert(expires_at <= event_timestamp);
21
22        const dr_account = self.get_account(p.debit_account_id);
23        assert(dr_account.debits_pending >= p.amount);
24
25        const cr_account = self.get_account(p.credit_account_id);
26        assert(cr_account.credits_pending >= p.amount);
27
28        // Pending transfers can expire in closed accounts.
29        assert(dr_account.flags.closed or !dr_account.flags.closed);
30        assert(cr_account.flags.closed or !cr_account.flags.closed);
31
32        // ...
33
34        const transfer_pending = self.get_transfer_pending(p.timestamp);
35        assert(p.timestamp == transfer_pending.timestamp);
36        assert(transfer_pending.status == .pending);
37        self.transfer_update_pending_status(&transfer_pending, .expired);
38    }
39
40    // ...
41 }
```

:ado

+ Testes + Incidentes

100

10² =

Tempo =

```
1 fn execute_expire_pending_transfers(self: *StateMachine, timestamp: u64) usize {
2     assert(timestamp > self.commit_timestamp);
3     assert(self.scan_lookup_results.items.len > 0);
4
5     // ....
6
7     assert(result_count <= result_max);
8     assert(self.scan_lookup_buffer_index > 0);
9     assert(self.scan_lookup_buffer_index == result_count * @sizeOf(Transfer));
10
11    for (transfers_pending, 0..) |*p, index| {
12        assert(p.flags.pending);
13        assert(p.timeout > 0);
14
15        const event_timestamp = timestamp - transfers_pending.len + index + 1;
16        assert(TimestampRange.valid(event_timestamp));
17        assert(self.commit_timestamp < event_timestamp);
18
19        const expires_at = p.timestamp + p.timeout_ns();
20        assert(expires_at <= event_timestamp);
21
22        const dr_account = self.get_account(p.debit_account_id);
23        assert(dr_account.debits.pending > p.amount);
```

precondições

100

10² =

Tempo =

```
1 fn execute_expire_pending_transfers(self: *StateMachine, timestamp: u64) usize {
2     assert(timestamp > self.commit_timestamp);
3     assert(self.scan_lookup_results.items.len > 0);
4
5     // ....
6
7     assert(result_count <= result_max);
8     assert(self.scan_lookup_buffer_index > 0);
9     assert(self.scan_lookup_buffer_index == result_count * @sizeOf(Transfer));
10
11    for (transfers_pending, 0..) |*p, index| {
12        assert(p.flags.pending);
13        assert(p.timeout > 0);
14
15        const event_timestamp = timestamp - transfers_pending.len + index + 1;
16        assert(TimestampRange.valid(event_timestamp));
17        assert(self.commit_timestamp < event_timestamp);
18
19        const expires_at = p.timestamp + p.timeout_ns();
20        assert(expires_at <= event_timestamp);
21
22        const dr_account = self.get_account(p.debit_account_id);
23        assert(dr_account.debits.pending > p.amount);
```

limits

100

10² =

Tempo = |

```
11  for (transfers_pending, 0..) |*p, index| {
12      assert(p.flags.pending);
13      assert(p.timeout > 0);
14
15      const event_timestamp = timestamp - transfers_pending.len + index + 1;
16      assert(TimestampRange.valid(event_timestamp));
17      assert(self.commit_timestamp < event_timestamp);
18
19      const expires_at = p.timestamp + p.timeout_ns();
20      assert(expires_at ≤ event_timestamp);
21
22      const dr_account = self.get_account(p.debit_account_id);
23      assert(dr_account.debits_pending ≥ p.amount);
24
25      const cr_account = self.get_account(p.credit_account_id);
26      assert(cr_account.credits_pending ≥ p.amount);
27
28      // Pending transfers can expire in closed accounts.
29      assert(dr_account.flags.closed or !dr_account.flags.closed);
30      assert(cr_account.flags.closed or !cr_account.flags.closed);
31
32      // ....
33
```

regras de negócio

100

10² =

Tempo = |

```
11     for (transfers_pending, 0..) |*p, index| {
12         assert(p.flags.pending);
13         assert(p.timeout > 0);
14
15         const event_timestamp = timestamp - transfers_pending.len + index + 1;
16         assert(TimestampRange.valid(event_timestamp));
17         assert(self.commit_timestamp < event_timestamp);
18
19         const expires_at = p.timestamp + p.timeout_ns();
20         assert(expires_at ≤ event_timestamp);
21
22         const dr_account = self.get_account(p.debit_account_id);
23         assert(dr_account.debits_pending ≥ p.amount);
24
25         const cr_account = self.get_account(p.credit_account_id);
26         assert(cr_account.credits_pending ≥ p.amount);
27
28         // Pending transfers can expire in closed accounts.
29         assert(dr_account.flags.closed or !dr_account.flags.closed);
30         assert(cr_account.flags.closed or !cr_account.flags.closed);
31
32         // ....
33
```

no-op!

100

10² =

Tempo =

```
20     assert(expires_at <= event_timestamp);
21
22     const dr_account = self.get_account(p.debit_account_id);
23     assert(dr_account.debits_pending >= p.amount);
24
25     const cr_account = self.get_account(p.credit_account_id);
26     assert(cr_account.credits_pending >= p.amount);
27
28     // Pending transfers can expire in closed accounts.
29     assert(dr_account.flags.closed || !dr_account.flags.closed);
30     assert(cr_account.flags.closed || !cr_account.flags.closed);
31
32     // ...
33
34     const transfer_pending = self.get_transfer_pending(p.timestamp);
35     assert(p.timestamp == transfer_pending.timestamp);
36     assert(transfer_pending.status == .pending);
37     self.transfer_update_pending_status(&transfer_pending, .expired);
38 }
39
40 // ...
41 }
```

ação

1000X

$10^2 = \text{Esforço}$

Tempo = Design +

```
1 fn execute_expire_pending_transfers(self: *StateMachine, timestamp: u64) usize {
2     assert(timestamp > self.commit_timestamp);
3     assert(self.scan_looking_for_expired_transfers);
4     assert(self.forest.grooves.transfers_pending.length() == 0);
5     // ...
6
7     assert(result_count <= 1);
8     assert(self.scan_looking_for_expired_transfers);
9     assert(self.scan_looking_for_expired_transfers);
10    for transfers_pending in self.forest.grooves.transfers_pending {
11        assert(transfers_pending.timestamp > self.commit_timestamp);
12        assert(transfers_pending.status == .pending);
13        assert(p.timestamp > transfers_pending.timestamp);
14        const event_timestamp = p.timestamp;
15        assert(TimestampRange::from(event_timestamp).contains(transfers_pending.timestamp));
16        assert(self.commit_timestamp < event_timestamp);
17        assert(self.commit_timestamp < p.timestamp);
18        assert(transfers_pending.expires_at == self.commit_timestamp);
19        assert(transfers_pending.expires_at < p.timestamp);
20        assert(transfers_pending.expires_at < event_timestamp);
21        const dr_account = self.get_dr_account(transfers_pending);
22        assert(dr_account.dust == 0);
23        assert(dr_account.dust == 0);
24        const cr_account = self.get_cr_account(transfers_pending);
25        assert(cr_account.balance == 0);
26        assert(cr_account.balance == 0);
27        // Pending transfer
28        assert(dr_account.fee == 0);
29        assert(dr_account.fee == 0);
30        assert(cr_account.fee == 0);
31        assert(cr_account.fee == 0);
32        // ...
33
34        const transfer_pending = self.get_transfer_pending(p.timestamp);
35        assert(p.timestamp == transfer_pending.timestamp);
36        assert(transfer_pending.status == .pending);
37        self.transfer_update_pending_status(&transfer_pending, .expired);
38    }
39    // ...
40 }
41 }
```

+ Incidentes
tudo outra vez!!!

1000X

$10^2 = \text{Esforço} \text{ vs Resultado}$

Tempo = Design + Desenvolvimento + Testes + Incidentes

+
Revisão



1000X

$10^2 = \text{Esforço} \text{ vs Resultado}$

Tempo = Design + Desenvolvimento + Testes + Incidentes

+
Revisão



1000X

$10^2 = \text{Esforço} \ vs \ \text{Resultado}$

Tempo = Design + Desenvolvimento + Testes + Incidentes

1000X

$10^2 = \text{Esforço} \text{ vs Resultado}$

Tempo = Design + Desenvolvimento + Testes + Incidentes



Federico Lorenzi

<https://youtu.be/P9nLS2reUOo>

1000X

$10^2 = \text{Esforço} \text{ vs Resultado}$

Tempo = Design + Desenvolvimento + Testes + Incidentes

Deterministic Simulation Testing

1000x

$10^2 = \text{Esforço} \text{ vs Resultado}$

Tempo = Design + Desenvolvimento + Testes + Incidentes

Deterministic Simulation Testing

100

$10^2 =$ [

Tempo = [



```
fn sum_two_numbers(a: int, b: int) int {
    return a + b;
}

fn test_sum_two_numbers() void {
    const sum = sum_two_numbers(1, 2);
    expect_equals(sum, 3);
}
```

Deterministic Simulation Testing

100

$10^2 = [$

Tempo = [

```
● ● ●  
fn sum_two_numbers(a: int, b: int) int {  
    return a + b;  
}  
  
fn test_sum_two_numbers() void {  
    const sum = sum_two_numbers(1, 2);  
    expect_equals(sum, 3);  
}
```

Deterministic ~~Simulation~~ Testing
comportamento esperado

100

$10^2 = 1$

Tempo = [

```
fn sum_two_numbers(a: int, b: int) int {
    const sum = a + b;
    assert(sum >= a);
    assert(sum >= b);

    return sum;
}

fn fuzz_sum_two_numbers() void {
    for (0..10_000) {
        const a = random_int();
        const b = random_int();
        const sum = sum_two_numbers(a, b);
        expect_equals(sum, a + b);
    }
}
```

100

10² = |

Tempo = [

```
fn sum_two_numbers(a: int, b: int) int {
    const sum = a + b;
    assert(sum >= a);
    assert(sum >= b);

    return sum;
}

fn fuzz_sum_two_numbers() void {
    for (0..10_000) {
        const a = random_int();
        const b = random_int();
        const sum = sum_two_numbers(a, b);
        expect_equals(sum, a + b);
    }
}
```

inputs
randômicos

100

10² = |

Tempo = [

```
fn sum_two_numbers(a: int, b: int) int {
    const sum = a + b;
    assert(sum >= a);
    assert(sum >= b);

    return sum;
}

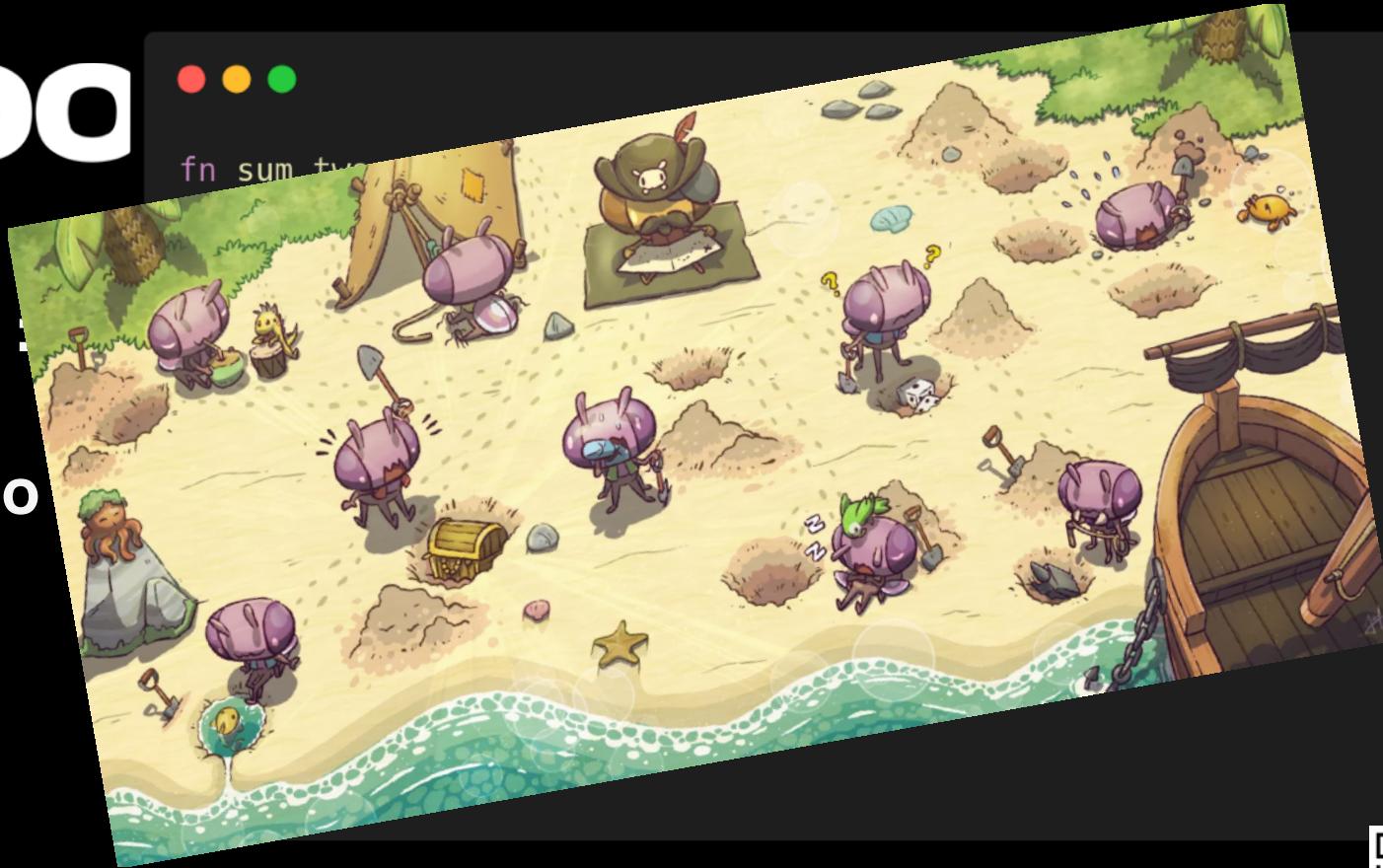
fn fuzz_sum_two_numbers() void {
    for (0..10_000) {
        const a = random_int();
        const b = random_int();
        const sum = sum_two_numbers(a, b);
        expect_equals(sum, a + b);
    }
}
```

mímica da
regra de negócio

100

10²

Tempo



100

$10^2 = 1$

Tempo = [

```
fn sum_two_numbers(a: int, b: int) int {
    const sum = a + b;
    assert(sum >= a);
    assert(sum >= b);

    return sum;
}
```

```
fn fuzz_sum_two_numbers() void {
    for (0..10_000) {
        const a = random_int();
        const b = random_int();
        const sum = sum_two_numbers(a, b);
        expect_equals(sum, a + b);
    }
}
```

overflow!

100

$10^2 = 1$

Tempo = [

```
fn sum_two_numbers(a: int, b: int) int {
    const sum = a + b;
    assert(sum >= a);
    assert(sum >= b);

    return sum;
}
```

```
fn fuzz_sum_two_numbers() void {
    for (0..10_000) {
        const a = random_int();
        const b = random_int();
        const sum = sum_two_numbers(a, b);
        expect_equals(sum, a + b);
    }
}
```

invariante

1000x

$10^2 = \text{Esforço} \text{ vs Resultado}$

Tempo = Design + Desenvolvimento + Testes + Incidentes

Deterministic **S**imulation **T**esting

100

10² = |

Tempo = [

```
> ./zig/zig build fuzz -- lsm_scan
info(fuzz): Fuzz seed = 2372797884104813212
info(lsm_scan_fuzz): commits = 986
info(lsm_scan_fuzz): query_specs[0]: (index_02 OR index_07 OR index_08 OR index_10) ascending
info(lsm_scan_fuzz): query_specs[1]: (index_01 AND index_11 AND (index_07 OR index_09 OR index_05 OR index_06) AND
index_08 AND (index_03 OR (index_04 AND index_13) OR index_10)) descending
info(lsm_scan_fuzz): query_specs[2]: (((index_06 OR index_09) AND index_04 AND index_02 AND index_08) OR (index_03
AND index_05) OR ((index_11 AND index_13) OR index_01 OR index_10)) ascending
info(lsm_scan_fuzz): query_specs[3]: (index_12 AND index_04 AND index_06 AND index_11 AND index_02) descending
info(lsm_scan_fuzz): query_specs[4]: ((index_08 OR index_01) AND index_03 AND (index_09 OR index_05 OR index_12 OR
index_04 OR index_06) AND index_02) ascending
info(lsm_scan_fuzz): query_specs[5]: (index_08 OR index_13 OR index_02 OR index_07 OR index_10) descending
info(lsm_scan_fuzz): query_specs[6]: (((index_10 OR (index_11 AND index_05 AND index_08) OR index_04 OR index_09) AND
(index_13 OR index_03) AND index_07 AND (index_12 AND index_06))) descending
info(lsm_scan_fuzz): query_specs[7]: (index_07 AND (index_09 OR (index_05 AND index_08 AND index_13) OR index_02 OR
index_01) AND index_12) descending
info(lsm_scan_fuzz): Passed!
info(fuzz): done in 30.456s
```

Deterministic Simulation Testing

100

10² = |

Tempo = [

```
> ./zig/zig build fuzz -- lsm_scan
info(fuzz): Fuzz seed = 2372797884104813212
info(lsm_scan_fuzz): commits = 986
info(lsm_scan_fuzz): query_specs[0]: (index_02 OR index_07 OR index_08 OR index_10) ascending
info(lsm_scan_fuzz): query_specs[1]: (index_01 AND index_11 AND (index_07 OR index_09 OR index_05 OR index_06) AND
index_08 AND (index_03 OR (index_04 AND index_13) OR index_10)) descending
info(lsm_scan_fuzz): query_specs[2]: (((index_06 OR index_09) AND index_04 AND index_02 AND index_08) OR (index_03
AND index_05) OR ((index_11 AND index_13) OR index_01 OR index_10)) ascending
info(lsm_scan_fuzz): query_specs[3]: (index_12 AND index_04 AND index_06 AND index_11 AND index_02) descending
info(lsm_scan_fuzz): query_specs[4]: ((index_08 OR index_01) AND index_03 AND (index_09 OR index_05 OR index_12 OR
index_04 OR index_06) AND index_02) ascending
info(lsm_scan_fuzz): query_specs[5]: (index_08 OR index_13 OR index_02 OR index_07 OR index_10) descending
info(lsm_scan_fuzz): query_specs[6]: (((index_10 OR (index_11 AND index_05 AND index_08) OR index_04 OR index_09) AND
(index_13 OR index_03) AND index_07 AND (index_12 AND index_06))) descending
info(lsm_scan_fuzz): query_specs[7]: (index_07 AND (index_09 OR (index_05 AND index_08 AND index_13) OR index_02 OR
index_01) AND index_12) descending
info(lsm_scan_fuzz): Passed!
info(fuzz): done in 30.456s
```

Deterministic Simulation Testing

condições válidas
aleatórias

1000X

$10^2 = \text{Esforço} \text{ vs Resultado}$

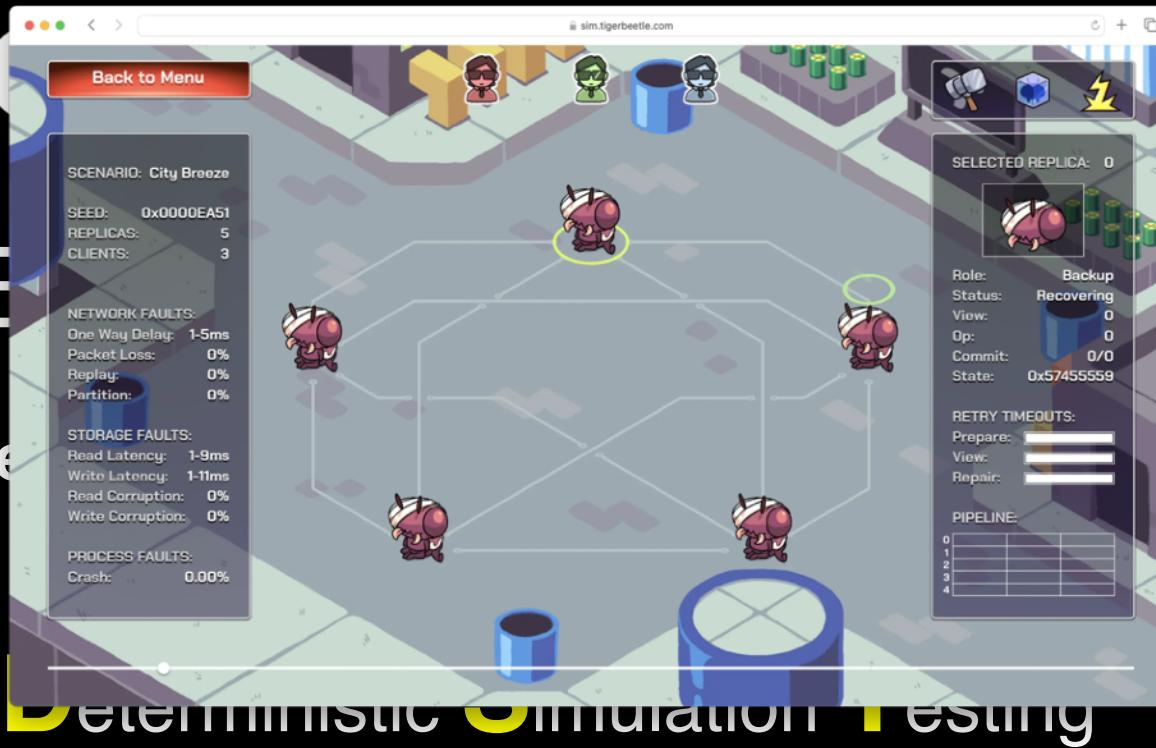
Tempo = Design + Desenvolvimento + Testes + Incidentes

Deterministic **S**imulation **T**esting

1000

$10^2 = E$

Tempo = Delay
Occurrences



Deterministic Simulation Testing

<https://sim.tigerbeetle.com>



1000X

$10^2 = \text{Esforço} \text{ vs Resultado}$

Tempo = Design + Desenvolvimento + Testes + Incidentes

Deterministic **S**imulation **T**esting

1000X

$10^2 = \text{Esforço} \text{ vs Resultado}$

Tempo = Design + Desenvolvimento + Testes + Incidentes

Deterministic **S**imulation **T**esting

1000X

$10^2 = \text{Esforço} \text{ vs Resultado}$

Tempo = Design + Desenvolvimento + Testes + Incidentes

1000x

$10^2 =$

Tempo =



1000X

$10^2 = \text{Esforço} \text{ vs Resultado}$

Tempo = Design + Desenvolvimento + Testes + Incidentes

The image shows a stack of four pages from a document titled "TigerBeetle 0.16.1" by JEPSEN. The top page is a summary or executive summary. The subsequent pages provide detailed technical information about the system's architecture, specifically regarding its replicated storage mechanism. The text discusses how TigerBeetle uses a replicated log to store data across multiple hosts, with each host maintaining a primary and backup copy. It also explains how the system handles replication between hosts and how it manages data consistency and durability.

JEPSEN

TigerBeetle 0.16.1

Viewstamped Replication

1 Background

TigerBeetle



1000x

10²

Tempo



s + Incidentes



1000X

$10^2 =$

Tempo = D



Incidentes



1000x

$10^2 =$



Tempo :

```
fn assert(ok: bool) void {
    if (!ok) 💀💀💀💀💀
```

1000X

$10^3 = \text{Build Trust and Have Fun}$

“Cultivar confiança e se divertir”



1000X

$10^3 = \text{Build Trust and Have Fun}$

“Cultivar confiança e se divertir”



1000X

$10^3 = \text{Build Trust and Have Fun}$

“Cultivar confiança e se divertir”



1000X

Muito obrigado!

- > Perguntas?
- > Comentários!

