

Developer Productivity at a Developer Productivity Startup



Robert-Jan ("RJ") Huijsman
Founding Engineer
reboot.dev

how do you guys get so much done with such a small team

1 Reply



reboot

reboot.dev

we focus on...

we focus on...
friction

we focus on...

friction

nondeterminism

while...

while...
coding

while...
coding
humaning

	friction	nondeterminism
humaning		
coding		

	friction	nondeterminism
humaning		
coding		

friction is...

anything that

slows you down

in getting to the next result

friction in humaning is...

"I can't start work on the high-priority bug until Monday's sprint planning has assigned the appropriate number of story points"

friction in humaning is...

"I need help but I'm too scared to ask"

but **most often...**

friction in **humaning** is...

"I can't get everyone together until Friday"

friction in **humaning** is...

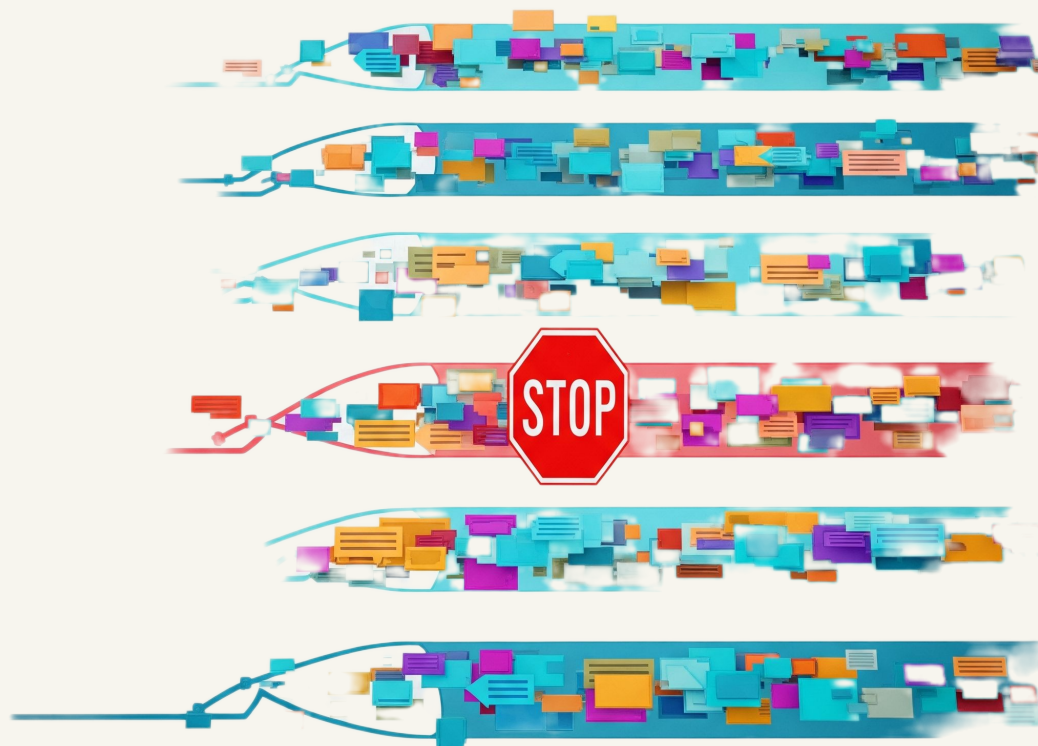
"I need input from Anna on that,
and she's a busy person"

friction in humaning is...

waiting
for other humans

throughput-focused
vs
latency-focused

throughput-focused
vs
latency-focused



1. pick up **new work** if you're fully blocked
2. use synchronous **overlap time** to unblock (if needed)

throughput-focused
vs
latency-focused

throughput-focused
vs
latency-focused





	friction	nondeterminism
humaning		
coding		

friction in coding is...

`"npm install` takes ages"

friction in coding is...

"where do I even make this change?"

friction in coding is...

[...]

frictionless coding is...

3. ???

2. ???

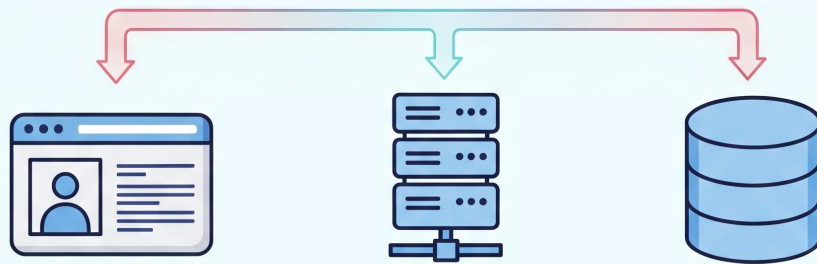
1. ???

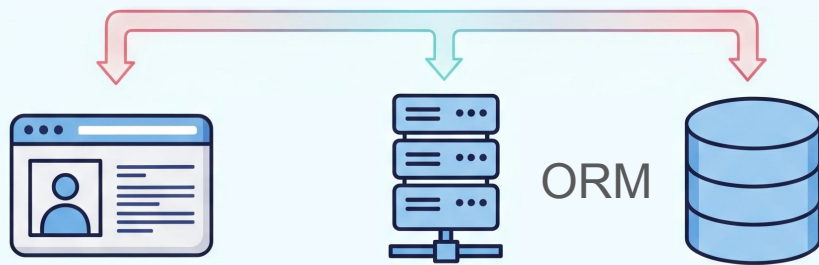
frictionless coding is...

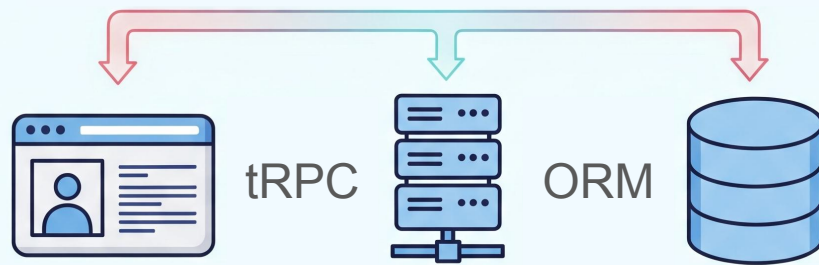
3. types – everywhere

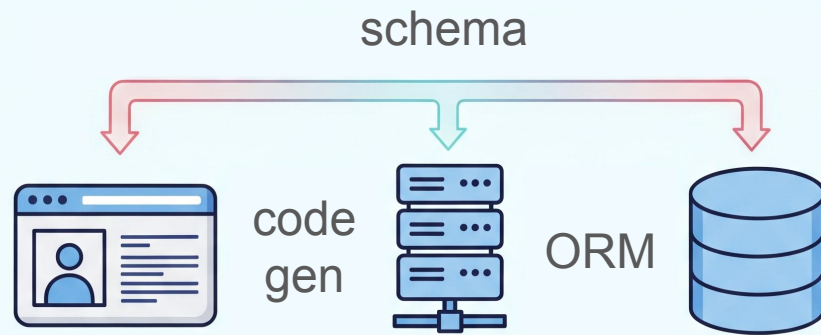
2. ???

1. ???



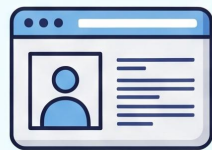








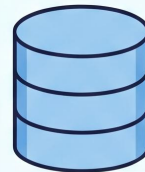
schema



code
gen



ORM

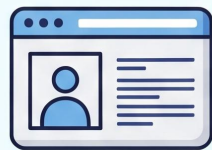


* Stainless ~~SQL~~Alchemy

speakeasy  Prisma



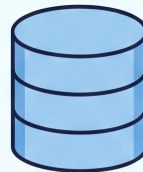
schema



code
gen



ORM



reboot

frictionless coding is...

3. types – everywhere

2. ???

1. ???

frictionless coding is...

- 3. types – everywhere
- 2. unit tests – for everything**
- 1. ???

frictionless coding is...

1. ???
2. unit tests everywhere
3. type checking everywhere



frictionless coding is...

3. types – everywhere

2. unit tests – for everything

1. stop repeating yourself – and others

```
const admin = require('firebase-admin');
const db = admin.firestore();

exports.getAccount = functions.https.onRequest(async (req, res) => {
  try {
    // Fetch a specific account by ID.
    const accountRef = db.collection('accounts').doc('accountId123');
    const state = await accountRef.get();

    // Check if the account actually exists.
    if (!state.exists) {
      res.status(404).send('Account not found');
      return;
    }

    // ... Business logic goes here ...

  } catch (error) {
    console.error('Error fetching account:', error);
    res.status(500).send('Error fetching account');
  }
});
```

orthogonal persistence

```
export const AccountServicer = Account.servicer({  
  deposit: async (  
    context: WriterContext,  
    state: Account.State, // NOTE: state is already here!  
    request: Account.DepositRequest  
  ) => {  
    // Business logic goes here; for example...  
    state.balance += request.amount;  
  },  
  
});
```

frictionless coding is...

3. types – everywhere
2. unit tests – for everything
1. stop repeating yourself – and others

frictionless coding is...

constant trade-offs

	friction	nondeterminism
humaning		
coding		

nondeterminism is...

anything that
makes success **unpredictable**

friction makes you go "aargh"

...

friction makes you go "aargh"
nondeterminism makes you quit

nondeterminism in humaning is...

"am I working on something useful?"

nondeterminism in humaning is...

"why did they assign this to me?"

nondeterminism in humaning is...

"I'm 60% confident my reviewer will like
this approach!"

Reboot: brainstorm



keep updating the **best idea**
get to **uncontroversial**
you own the progress,
you own the problem,
you **don't own** the idea that solves it

	friction	nondeterminism
humaning		
coding		

nondeterminism in coding is...

"my tests are flaky"

nondeterminism in coding is...

"it went wrong that one time in production, we don't know why"

nondeterminism in coding is...

"I finally found that race condition!"

nondeterminism in coding is...

the most painful bugs a developer can be
asked to solve

three steps to **eliminate**
nondeterminic behavior:

3. ???

2. ???

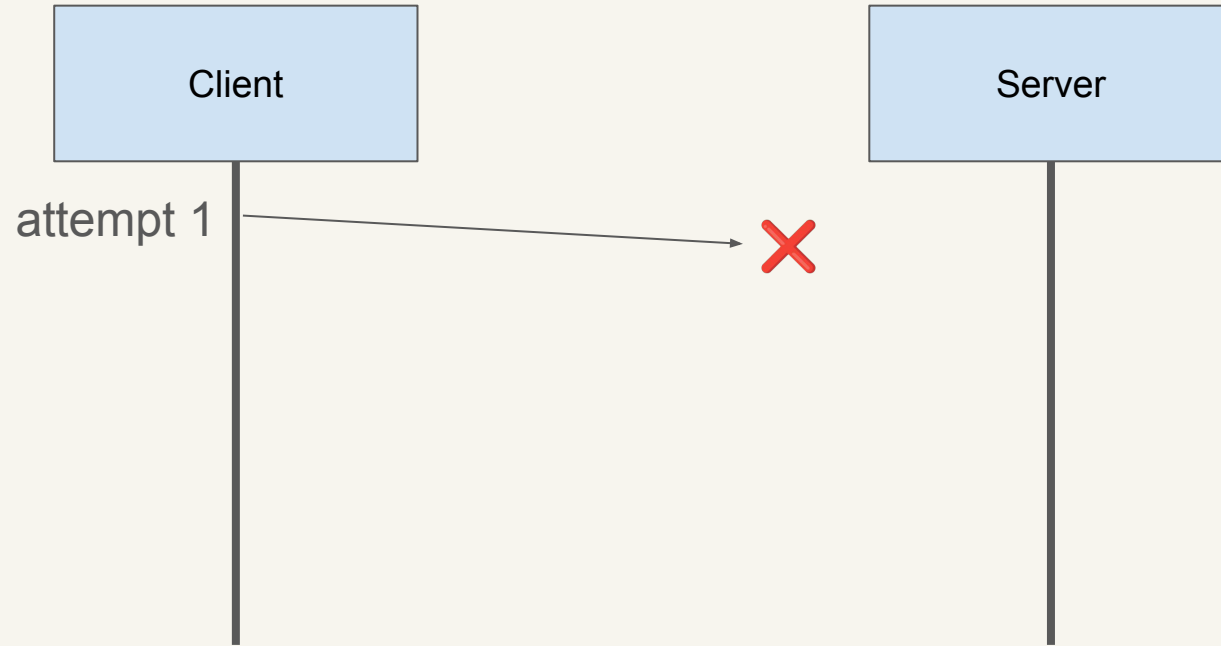
1. ???

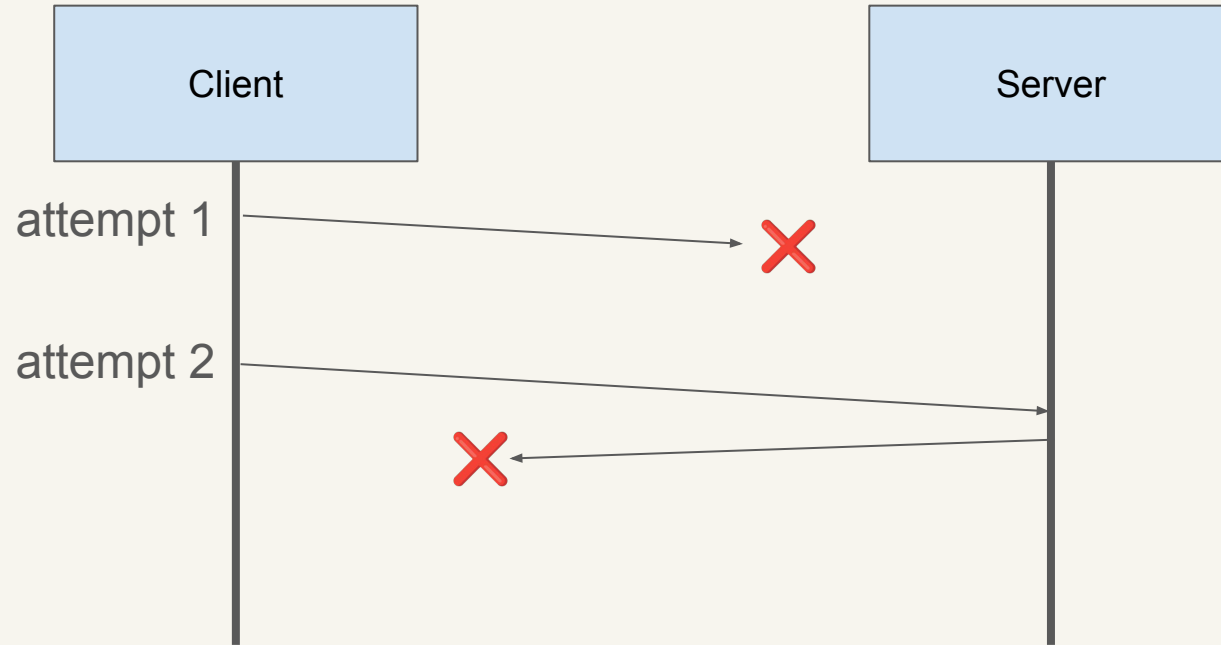
three steps to eliminate
nondeterminic behavior:

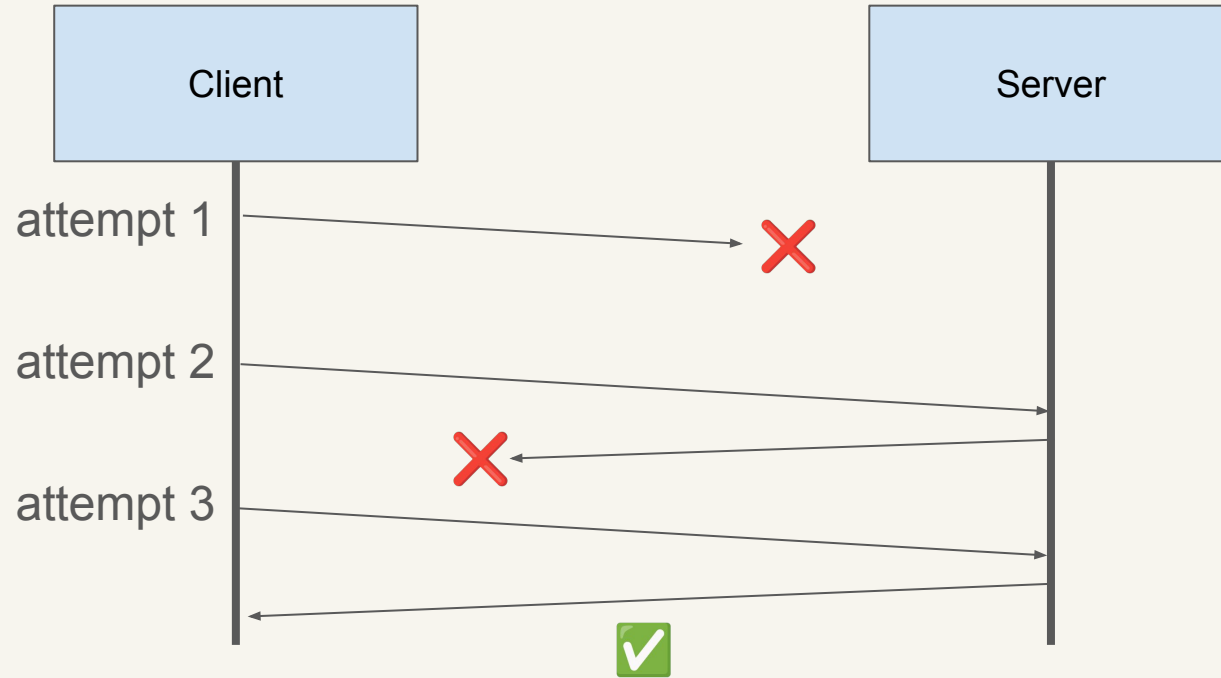
3. retries and idempotency

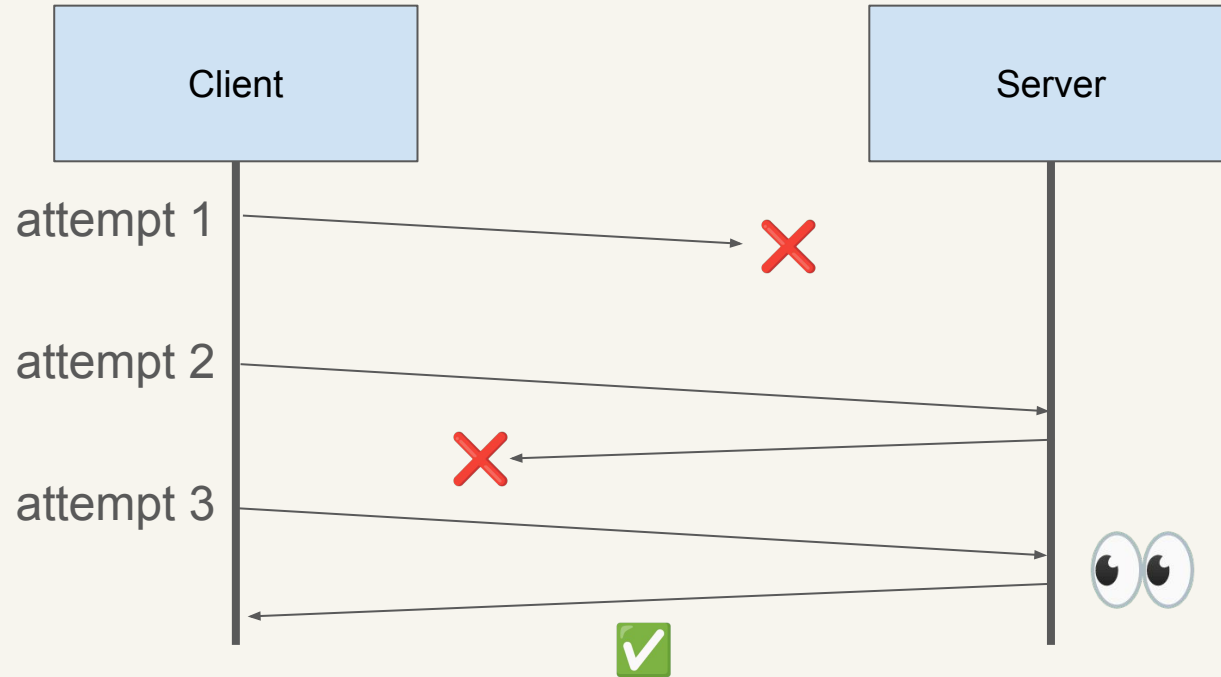
2. ???

1. ???









three steps to eliminate
nondeterminic behavior:

3. retries and idempotency

2. ???

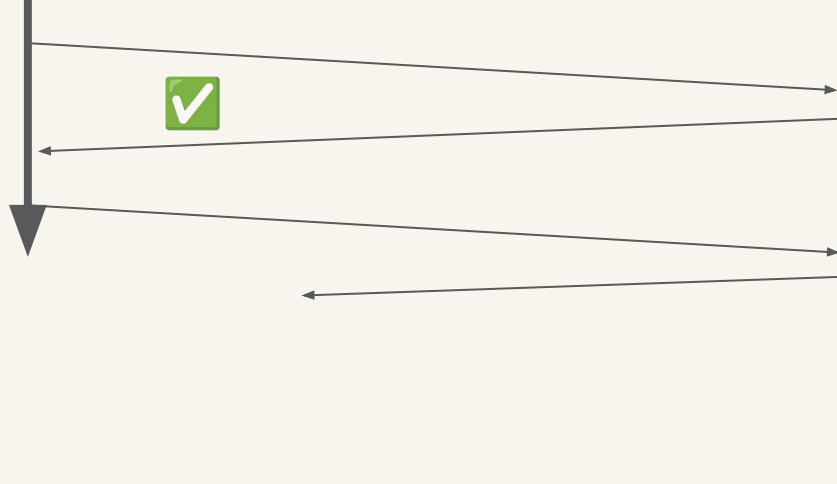
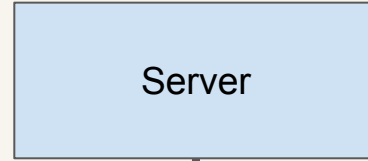
1. ???

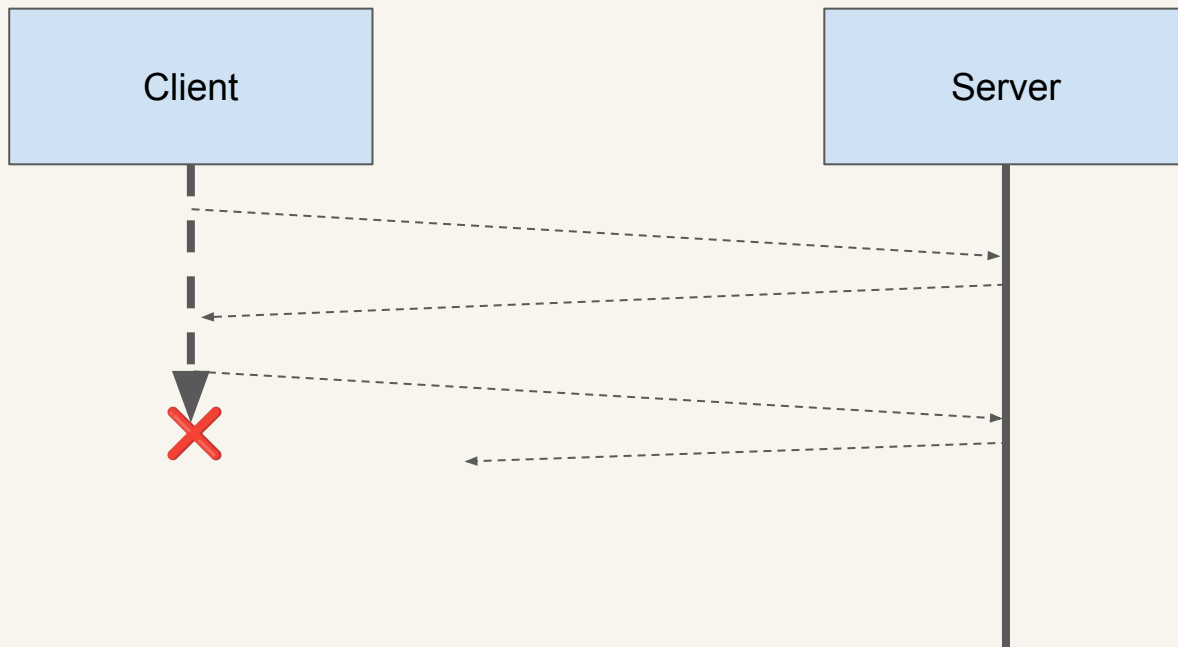
three steps to eliminate
nondeterminic behavior:

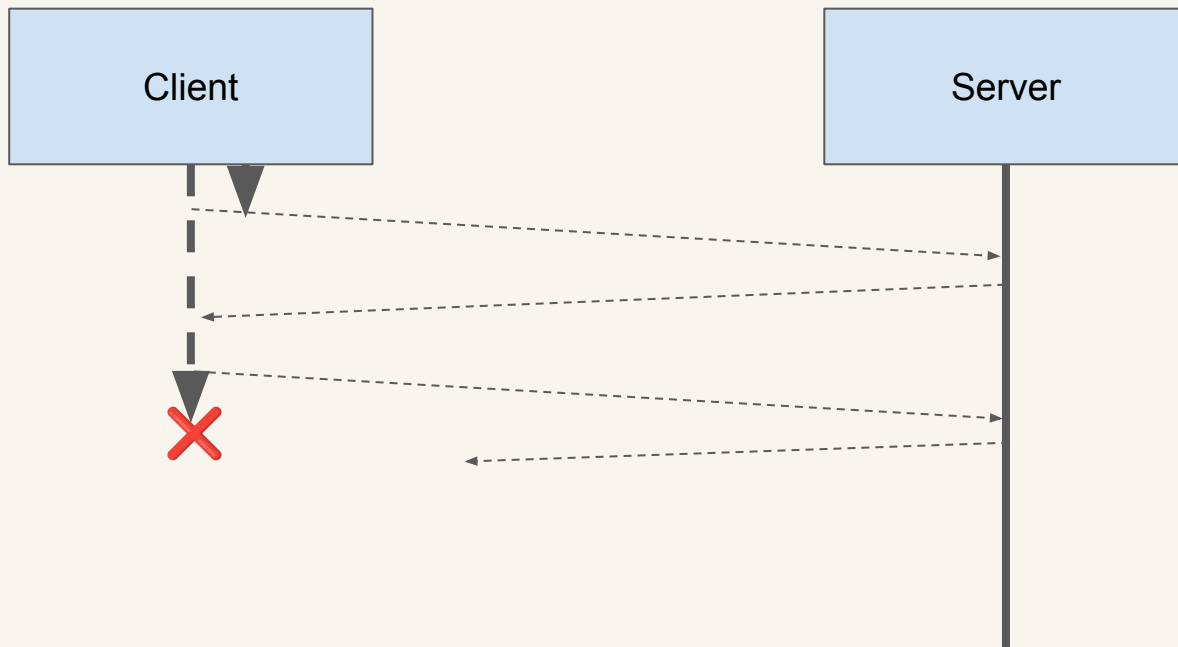
3. retries and idempotency

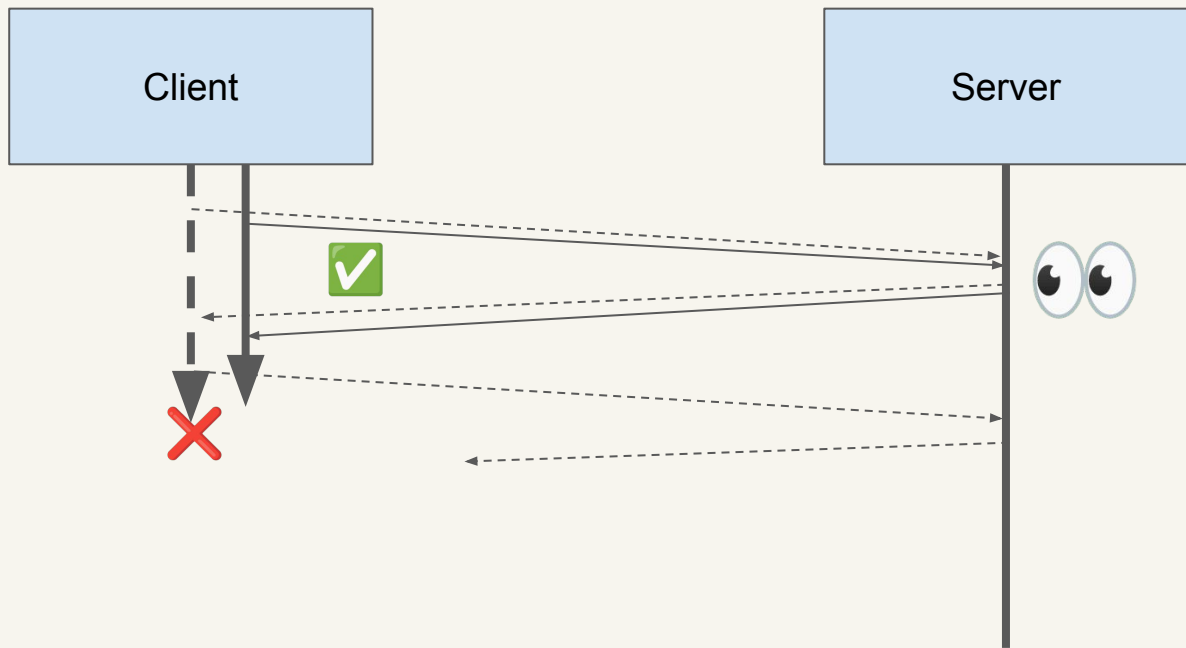
2. durable execution

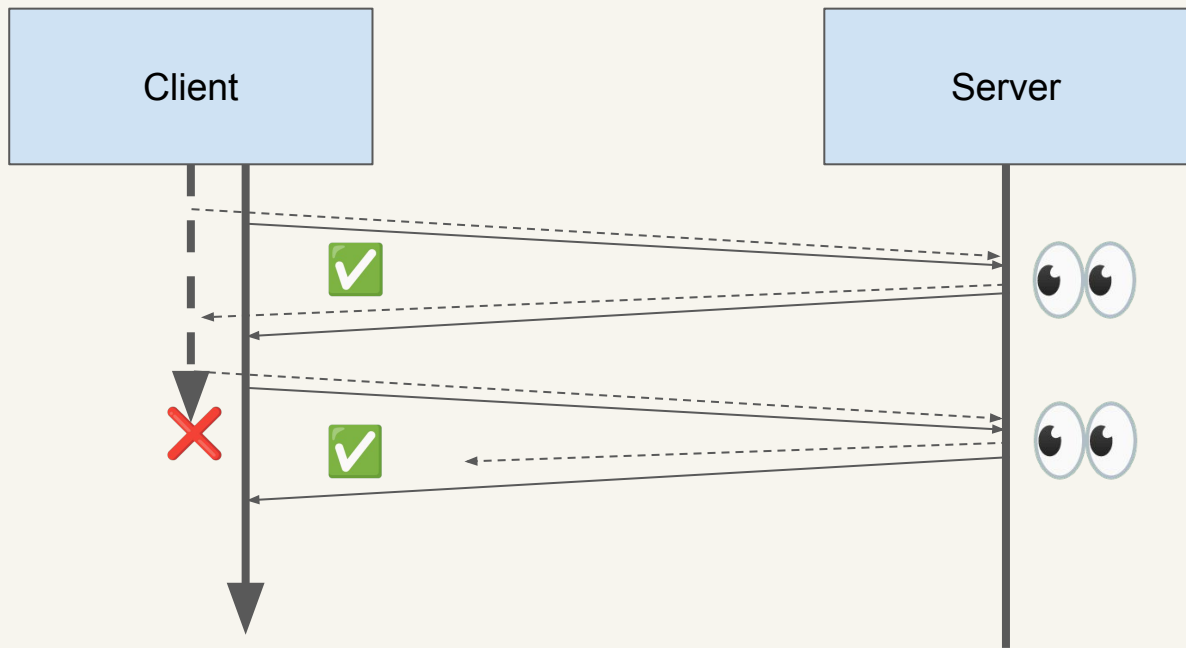
1. ???









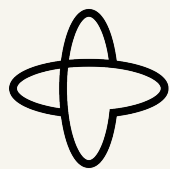


durable execution



no work unexpectedly left undone

durable execution: frameworks



Temporal



restate

reboot

three steps to eliminate
nondeterminic behavior:

3. retries and idempotency

2. durable execution

1. ???

three steps to eliminate
nondeterminic behavior:

3. retries and idempotency
2. durable execution
- 1. strong consistency**

account ID	balance
01	DKK 1000
23	DKK 2000
45	DKK 3000
67	DKK 4000
89	DKK 5000



account ID	balance
01	DKK ~~4000~~ 1500
23	DKK ~~2000~~ 1500
45	DKK 3000
67	DKK 4000
89	DKK 5000



account ID	balance
01	DKK ~~1000~~ 1500
23	DKK ~~2000~~ 1500
45	DKK 3000
67	DKK 4000
89	DKK 5000



	account ID	balance	
	01	DKK ~~1000~~ 1500	
	23	DKK ~~2000~~ 1500	
	45	DKK 3000	
	67	DKK ~~4000~~ 3000	
	89	DKK ~~5000~~ 6000	

the three reasons
you don't have strong consistency:

- your database
- your code layout
- your mistake

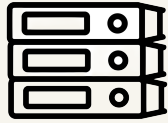
the three reasons
you don't have strong consistency:

- **your database**
- your code layout
 - your mistake

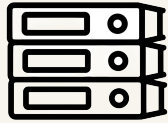
the three reasons
you don't have strong consistency:

- your database
- **your code layout**
 - your mistake

transfer()

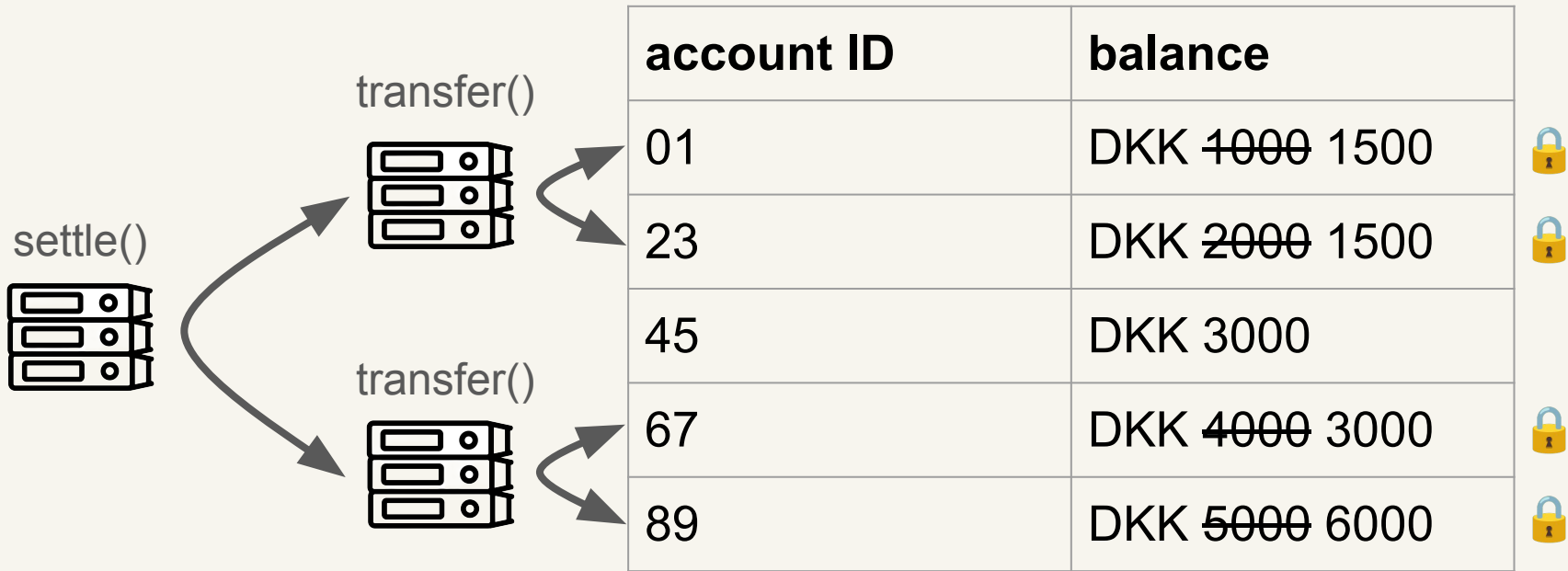


transfer()



account ID	balance
01	DKK ~~4000~~ 1500
23	DKK ~~2000~~ 1500
45	DKK 3000
67	DKK ~~4000~~ 3000
89	DKK ~~5000~~ 6000





the three reasons
you don't have strong consistency:

- your database
- **your code layout**
 - your mistake

the three reasons
you don't have strong consistency:

- your database
- your code layout
- **your mistake**

```
const admin = require('firebase-admin');
const db = admin.firestore();

exports.transferCredits = functions.https.onRequest(async (req, res) => {
  const { fromAccountId, toAccountId, amount } = req.body;

  try {
    const result = await db.runTransaction(async (transaction) => {
      // Get references to both accounts.
      const fromAccountRef = db.collection('accounts').doc(fromAccountId);
      const toAccountRef = db.collection('accounts').doc(toAccountId);

      // Read both accounts.
      const fromAccount = await fromAccountRef.get();
      const toAccount = await toAccountRef.get();

      // ...
    });
  }
});
```

```
const admin = require('firebase-admin');
const db = admin.firestore();

exports.transferCredits = functions.https.onRequest(async (req, res) => {
  const { fromAccountId, toAccountId, amount } = req.body;

  try {
    const result = await db.runTransaction(async (transaction) => {
      // Get references to both accounts.
      const fromAccountRef = db.collection('accounts').doc(fromAccountId);
      const toAccountRef = db.collection('accounts').doc(toAccountId);

      // Read both accounts.
      const fromAccount = await fromAccountRef.get();
      const toAccount = await toAccountRef.get();

      // ...
    });
  }
});
```

```
const admin = require('firebase-admin');
const db = admin.firestore();

exports.transferCredits = functions.https.onRequest(async (req, res) => {
  const { fromAccountId, toAccountId, amount } = req.body;

  try {
    const result = await db.runTransaction(async (transaction) => {
      // Get references to both accounts.
      const fromAccountRef = db.collection('accounts').doc(fromAccountId);
      const toAccountRef = db.collection('accounts').doc(toAccountId);

      // Read both accounts.
      const fromAccount = await fromAccountRef.get();
      const toAccount = await toAccountRef.get();

      // ...
    });
  }
});
```

```
const admin = require('firebase-admin');
const db = admin.firestore();

exports.transferCredits = functions.https.onRequest(async (req, res) => {
  const { fromAccountId, toAccountId, amount } = req.body;

  try {
    const result = await db.runTransaction(async (transaction) => {
      // Get references to both accounts.
      const fromAccountRef = db.collection('accounts').doc(fromAccountId);
      const toAccountRef = db.collection('accounts').doc(toAccountId);

      // Read both accounts within the transaction.
      const fromAccount = await transaction.get(fromAccountRef);
      const toAccount = await transaction.get(toAccountRef);

      // ...
    });
  }
});
```

```
const admin = require('firebase-admin');
const db = admin.firestore();

exports.transferCredits = functions.https.onRequest(async (req, res) => {
  const { fromAccountId, toAccountId, amount } = req.body;

  try {
    const result = await db.runTransaction(async (transaction) => {
      // Get references to both accounts
      const fromAccountRef = db.collection('accounts').doc(fromAccountId);
      const toAccountRef = db.collection('accounts').doc(toAccountId);

      // Read both accounts with the transaction.
      const fromAccount = await transaction.get(fromAccountRef);
      const toAccount = await transaction.get(toAccountRef);

      // ...
    });
  }
});
```



the three reasons
you don't have strong consistency:

- your database
- your code layout
 - your mistake

```
export const AccountServicer = Account.servicer({  
  transfer: async (  
    context: TransactionContext, // NOTE: transaction!  
    state: Account.State,  
    request: Account.TransferRequest  
  ) => {  
    state.balance -= request.amount;  
    await Account.ref(request.toAccountId).deposit(  
      context,  
      amount=request.amount  
    )  
  },  
});
```

three steps to **eliminate**
nondeterminic behavior:

3. retries and idempotency
2. durable execution
1. strong consistency

three steps to **eliminate** nondeterminic behavior:

3. retries and idempotency

the work will start

2. durable execution

the work will complete

1. strong consistency

the work will be correct

	friction	nondeterminism
humaning		
coding		

how do you guys get so much done with such a small team

1 Reply



async workstreams

types everywhere
unit tests for everything
intolerance to repetition

brainstorms

retries & idempotency
durable execution
strong consistency

and they developed
happily ever after
...?

nope

friction we have today

- GitHub Codespaces have disappointing filesystem performance
 - ... saving files can take ~20 seconds!?!
- Bazel takes O(minutes) to start up the first time
 - ... possibly due to that filesystem performance issues
 - ... but also our workspace is just really big
- Adding new Bazel dependencies is super painful due to versioning conflicts
 - ... we really should migrate to bzlmod!
- Our integration tests are flaky due to CPU starvation leading to timeouts
- LLMs frequently get confused by the concept of generated code
- Our generated code can get so large that it's hard to read
- Some unit-tests require that you build a `.whl` distribution, which takes a minute.
- Our Python <-> TypeScript connector contains a lot of boilerplate code.
- We're lacking good documentation for some of our generated APIs
- Live-reloading an application can take longer than we'd like
- It can be hard to find readable documentation on past brainstorm outcomes
- Our docker containers are larger than we'd like, so they're slow to push
- LLMs still fail to help me organize a stack of commits properly
- ...

friction is a never-ending fight

but ...

**beating nondeterminism
is a one-time investment**

make the **investment**
or suffer the **consequences**

make t tment
or suffer t equences



make the investment
or **our future career** will be
to suffer **your consequences**



[linkedin.com/in/rjhuijsman](https://www.linkedin.com/in/rjhuijsman)

reboot

reboot.dev



discord.gg/cRbdcS94Nr