

# Information Theory vs Filters

# **Intro**

# Lukáš Hozda

- Marketing at Braiins

# Lukáš Hozda

- Marketing at Braiins
- Rust/Lisp programmer

# Lukáš Hozda

- Marketing at Braiins
- Rust/Lisp programmer
- Bitcoin, Emacs, HTMX, Linux enthusiast

# Lukáš Hozda

- Marketing at Braiins
- Rust/Lisp programmer
- Bitcoin, Emacs, HTMX, Linux enthusiast
- Teaching Rust at MatFyz



# Building bitcoin in Rust

Written by Lukáš Hozda

With contributors: Daniela Brozzoni, Luukas Pörtfors

# Context

- There is an ongoing Core vs Knots debate

# Context

- There is an ongoing Core vs Knots debate
- Filtering is one of the main (technical) fault lines

# Context

- There is an ongoing Core vs Knots debate
- Filtering is one of the main (technical) fault lines
  - ▶ Proponents of filtering want to filter arbitrary data embedding

# Context

- There is an ongoing Core vs Knots debate
- Filtering is one of the main (technical) fault lines
  - ▶ Proponents of filtering want to filter arbitrary data embedding
  - ▶ All non-monetary data on the blockchain == spam (?) and should be rejected

# Context

- There is an ongoing Core vs Knots debate
- Filtering is one of the main (technical) fault lines
  - Proponents of filtering want to filter arbitrary data embedding
  - All non-monetary data on the blockchain == spam (?) and should be rejected
- Focus today: mempool / relay-level filtering

# Context

- There is an ongoing Core vs Knots debate
- Filtering is one of the main (technical) fault lines
  - Proponents of filtering want to filter arbitrary data embedding
  - All non-monetary data on the blockchain == spam (?) and should be rejected
- Focus today: mempool / relay-level filtering
- BIP-110 exists, but consensus-level filtering is out of scope today

# Ordinals

- Ordinals are a convention for tracking individual satoshis

# Ordinals

- Ordinals are a convention for tracking individual satoshis
- More precisely, ordinal theory is one ruleset for saying “these sats are the same” across transactions

# Ordinals

- Ordinals are a convention for tracking individual satoshis
- More precisely, ordinal theory is one ruleset for saying “these sats are the same” across transactions
- You could define a different tracking ruleset; the point is social coordination and shared interpretation

# Ordinals

- Ordinals are a convention for tracking individual satoshis
- More precisely, ordinal theory is one ruleset for saying “these sats are the same” across transactions
- You could define a different tracking ruleset; the point is social coordination and shared interpretation
- They assign each sat a stable identity/index within the total supply

# Ordinals

- Ordinals are a convention for tracking individual satoshis
- More precisely, ordinal theory is one ruleset for saying “these sats are the same” across transactions
- You could define a different tracking ruleset; the point is social coordination and shared interpretation
- They assign each sat a stable identity/index within the total supply
- That lets people say “this specific sat moved here”

# Ordinals

- Ordinals are a convention for tracking individual satoshis
- More precisely, ordinal theory is one ruleset for saying “these sats are the same” across transactions
- You could define a different tracking ruleset; the point is social coordination and shared interpretation
- They assign each sat a stable identity/index within the total supply
- That lets people say “this specific sat moved here”
- By themselves, ordinals are about identification and tracking, not content

# Inscriptions

- Inscriptions are arbitrary data embedded in Bitcoin transactions (usually witness data)

# Inscriptions

- Inscriptions are arbitrary data embedded in Bitcoin transactions (usually witness data)
  - ▶ Witness data is discounted, so they can fit in more of it
- In the ordinals ecosystem, an inscription is associated with a specific sat

# Inscriptions

- Inscriptions are arbitrary data embedded in Bitcoin transactions (usually witness data)
  - ▶ Witness data is discounted, so they can fit in more of it
- In the ordinals ecosystem, an inscription is associated with a specific sat
- The payload can be images, text, HTML, or other bytes

# Inscriptions

- Inscriptions are arbitrary data embedded in Bitcoin transactions (usually witness data)
  - ▶ Witness data is discounted, so they can fit in more of it
- In the ordinals ecosystem, an inscription is associated with a specific sat
- The payload can be images, text, HTML, or other bytes
- This is the part that usually triggers filtering debates

# Difference

- Ordinal = which sat

# Difference

- **Ordinal** = which sat
- **Inscription** = what data is attached/associated

# Difference

- **Ordinal** = which sat
- **Inscription** = what data is attached/associated
- You can discuss ordinals as tracking without discussing inscriptions

# Difference

- **Ordinal** = which sat
- **Inscription** = what data is attached/associated
- You can discuss ordinals as tracking without discussing inscriptions
- In practice, people usually discuss the combined ordinals+inscriptions ecosystem

# Criticism

- Most criticism targets inscriptions and transaction patterns, not the numbering convention itself

# Criticism

- Most criticism targets inscriptions and transaction patterns, not the numbering convention itself
- Block weight is bounded, so this is not “infinite blockchain bloat per block”

# Criticism

- Most criticism targets inscriptions and transaction patterns, not the numbering convention itself
- Block weight is bounded, so this is not “infinite blockchain bloat per block”
  - ▶ The “unrestricted growth” is also often cited for OP\_RETURN

# Criticism

- Most criticism targets inscriptions and transaction patterns, not the numbering convention itself
- Block weight is bounded, so this is not “infinite blockchain bloat per block”
  - The “unrestricted growth” is also often cited for OP\_RETURN
- One concern is competition for scarce blockspace (including witness-heavy payloads)

# Criticism

- Most criticism targets inscriptions and transaction patterns, not the numbering convention itself
- Block weight is bounded, so this is not “infinite blockchain bloat per block”
  - The “unrestricted growth” is also often cited for OP\_RETURN
- One concern is competition for scarce blockspace (including witness-heavy payloads)
- Another concern is UTXO set growth from certain usage patterns

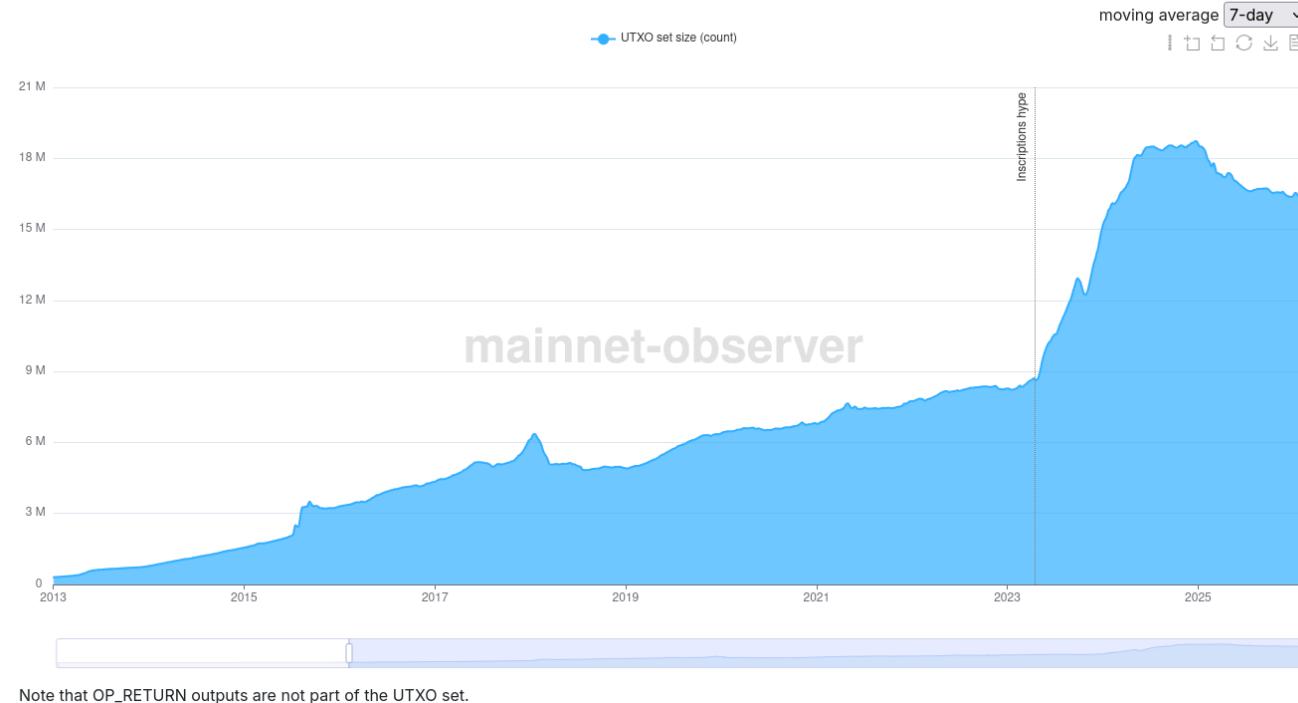
# Criticism

- Most criticism targets inscriptions and transaction patterns, not the numbering convention itself
- Block weight is bounded, so this is not “infinite blockchain bloat per block”
  - The “unrestricted growth” is also often cited for OP\_RETURN
- One concern is competition for scarce blockspace (including witness-heavy payloads)
- Another concern is UTXO set growth from certain usage patterns
- These are different resource problems and should not be conflated

# UTXO Set

## UTXO count

Shows the UTXO set size over time.



# Chain Growth

- The more precise concern is sustained chain growth over time and who uses scarce blockspace

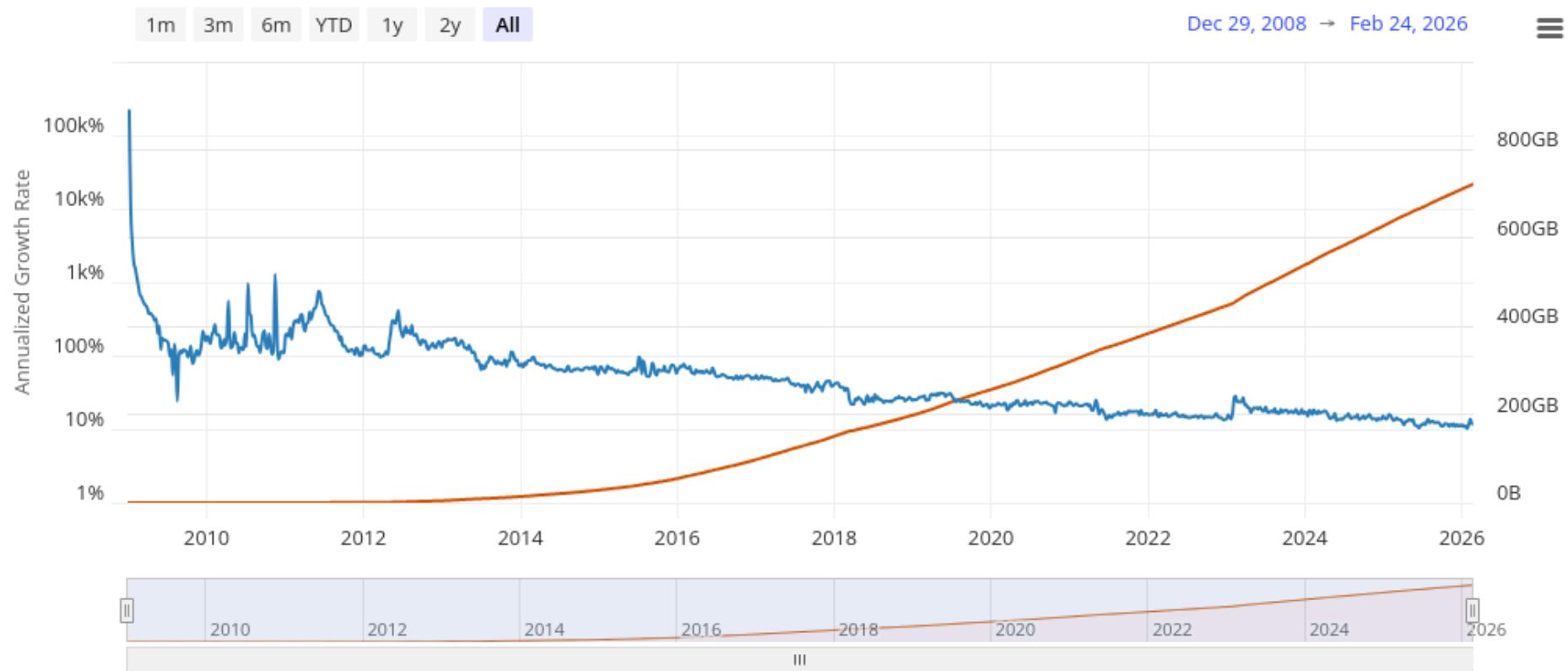
# Chain Growth

- The more precise concern is sustained chain growth over time and who uses scarce blockspace
- Historical storage, bandwidth, and validation costs still accumulate

# Chain Growth

- The more precise concern is sustained chain growth over time and who uses scarce blockspace
- Historical storage, bandwidth, and validation costs still accumulate
- UTXO set can shrink, the blockchain can't

## Intro



# **Network**

# Mempool

- There is no single global mempool

# Mempool

- There is no single global mempool
- Every node has its own local mempool

# Mempool

- There is no single global mempool
- Every node has its own local mempool
- Local policy decides what is accepted and relayed

# Mempool

- There is no single global mempool
- Every node has its own local mempool
- Local policy decides what is accepted and relayed
- Miners build blocks from what they see and what pays

# Gossip

- Nodes talk to peers, not to a central coordinator

# Gossip

- Nodes talk to peers, not to a central coordinator
- Transactions propagate hop by hop

# Gossip

- Nodes talk to peers, not to a central coordinator
- Transactions propagate hop by hop
- Topology matters: peers, connectivity, implementation mix

# Gossip

- Nodes talk to peers, not to a central coordinator
- Transactions propagate hop by hop
- Topology matters: peers, connectivity, implementation mix
- Small relay minorities can still create viable paths

# BTC wire protocol

- The protocol through which nodes talk is pretty simple
- We can make a relay just by being able to process a few message types

# Handshake

- **version**: announces protocol version, services, user agent, and chain height

# Handshake

- **version**: announces protocol version, services, user agent, and chain height
- **verack**: confirms the handshake

# Handshake

- `version`: announces protocol version, services, user agent, and chain height
- `verack`: confirms the handshake
- `ping`: keepalive / liveness check

# Handshake

- `version`: announces protocol version, services, user agent, and chain height
- `verack`: confirms the handshake
- `ping`: keepalive / liveness check
- `pong`: response to `ping`

# Relay

- `inv`: “I have object(s) with these hashes” (announcement only)

# Relay

- `inv`: “I have object(s) with these hashes” (announcement only)
- `getdata`: “send me the full object for these hashes”

# Relay

- `inv`: “I have object(s) with these hashes” (announcement only)
- `getdata`: “send me the full object for these hashes”
- `tx`: full transaction payload

# Discovery

- `getaddr`: “send me peer addresses”

# Discovery

- `getaddr`: “send me peer addresses”
- `addr`: list of peer addresses

# Relay Flow

- Node A sends `inv` with tx hash(es)

# Relay Flow

- Node A sends `inv` with tx hash(es)
- Node B decides what it wants

# Relay Flow

- Node A sends `inv` with tx hash(es)
- Node B decides what it wants
- Node B sends `getdata` for selected txs

# Relay Flow

- Node A sends `inv` with tx hash(es)
- Node B decides what it wants
- Node B sends `getdata` for selected txs
- Node A sends tx with full bytes

# Relay Flow

- Node A sends `inv` with tx hash(es)
- Node B decides what it wants
- Node B sends `getdata` for selected txs
- Node A sends tx with full bytes
- This reduces bandwidth compared to pushing full txs to everyone

# Filtering

- Mempool filtering is local policy, not consensus

# Filtering

- Mempool filtering is local policy, not consensus
- It can block relay through that node

# Filtering

- Mempool filtering is local policy, not consensus
- It can block relay through that node
- It cannot directly control the whole network topology

# Filtering

- Mempool filtering is local policy, not consensus
- It can block relay through that node
- It cannot directly control the whole network topology
- The real question is network-wide effectiveness

# Limits

# Topology

- Relay is path-based

# Topology

- Relay is path-based
- Transactions need some permissive paths, not universal approval

# Topology

- Relay is path-based
- Transactions need some permissive paths, not universal approval
- More connectivity makes bypass easier

# Topology

- Relay is path-based
- Transactions need some permissive paths, not universal approval
- More connectivity makes bypass easier
- Strong suppression needs near-universal adoption

## **Sub-1 sat/vB**

- We already have a concrete precedent

## Sub-1 sat/vB

- We already have a concrete precedent
- Many nodes reject or deprioritize sub-1 sat/vB transactions

## Sub-1 sat/vB

- We already have a concrete precedent
- Many nodes reject or deprioritize sub-1 sat/vB transactions
- They still propagate and still get mined

## Sub-1 sat/vB

- We already have a concrete precedent
- Many nodes reject or deprioritize sub-1 sat/vB transactions
- They still propagate and still get mined
- Relay policy does not equal miner/pool inclusion policy

## Sub-1 sat/vB

- We already have a concrete precedent
- Many nodes reject or deprioritize sub-1 sat/vB transactions
- They still propagate and still get mined
- Relay policy does not equal miner/pool inclusion policy
- Partial filtering is not network-wide suppression

# Steganography

- Steganography = hiding a message inside another valid-looking carrier

# Steganography

- Steganography = hiding a message inside another valid-looking carrier
- The receiver needs an extraction rule, but outsiders may only see an ordinary object

# Steganography

- Steganography = hiding a message inside another valid-looking carrier
- The receiver needs an extraction rule, but outsiders may only see an ordinary object
- The carrier still “works” for its normal purpose

# Steganography

- Steganography = hiding a message inside another valid-looking carrier
- The receiver needs an extraction rule, but outsiders may only see an ordinary object
- The carrier still “works” for its normal purpose
- Steganography is about covert encoding, not encryption

# Bitcoin

- Bitcoin transactions are structured, expressive, and highly constrained at the same time

# Bitcoin

- Bitcoin transactions are structured, expressive, and highly constrained at the same time
- Many different valid transactions can represent similar economic intent

# Bitcoin

- Bitcoin transactions are structured, expressive, and highly constrained at the same time
- Many different valid transactions can represent similar economic intent
- This gives adversaries room to encode side-information while staying consensus-valid

# Bitcoin

- Bitcoin transactions are structured, expressive, and highly constrained at the same time
- Many different valid transactions can represent similar economic intent
- This gives adversaries room to encode side-information while staying consensus-valid
- Filters must infer intent from valid transactions

# General Examples

- Image pixels: least-significant bits can carry a hidden message with little visible change

# General Examples

- Image pixels: least-significant bits can carry a hidden message with little visible change
- Text: whitespace, capitalization, or punctuation patterns can encode bits

# General Examples

- Image pixels: least-significant bits can carry a hidden message with little visible change
- Text: whitespace, capitalization, or punctuation patterns can encode bits
- Network traffic: timing, padding, or packet ordering can carry side-information

# General Examples

- Image pixels: least-significant bits can carry a hidden message with little visible change
- Text: whitespace, capitalization, or punctuation patterns can encode bits
- Network traffic: timing, padding, or packet ordering can carry side-information
- The same message can often move between multiple carriers

# Rust Toy

- I have a tiny Rust example that hides a message in image pixel bytes (LSB)

## Limits

## Rust Toy

- I have a tiny Rust example that hides a message in image pixel bytes (LSB)
- PNG works on the same principle after decoding to pixel bytes

## Limits

# Rust Toy

- I have a tiny Rust example that hides a message in image pixel bytes (LSB)
- PNG works on the same principle after decoding to pixel bytes
- I can show it on a BMP, and the encoding/decoding logic is the same

```
fn put_bit(byte: &mut u8, bit: u8) {  
    *byte = (*byte & !1) | (bit & 1);  
}  
  
for (i, bit) in bits.enumerate() {  
    put_bit(&mut pixels[i], bit);
```

```
}
```

```
let mut out = 0u8;
for (shift, b) in pixels[i..i + 8].iter().enumerate() {
    out |= (b & 1) << shift;
}
```

Limits

# Original BMP

## Limits



# Encoded BMP

## Limits



# BTC Examples

- Explicit payload fields (for example `OP_RETURN`)

# BTC Examples

- Explicit payload fields (for example `OP_RETURN`)
- Witness data payloads (the inscriptions case)

# BTC Examples

- Explicit payload fields (for example `OP_RETURN`)
- Witness data payloads (the inscriptions case)
- Data encoded indirectly via transaction structure/pattern choices

# BTC Examples

- Explicit payload fields (for example `OP_RETURN`)
- Witness data payloads (the inscriptions case)
- Data encoded indirectly via transaction structure/pattern choices
- The same information can be moved between multiple valid representations

# Encodings

- OP\_RETURN and visible witness payloads are easy to identify

# Encodings

- `OP_RETURN` and visible witness payloads are easy to identify
- Other encodings can be much less visible at policy level

# Encodings

- `OP_RETURN` and visible witness payloads are easy to identify
- Other encodings can be much less visible at policy level
- Example: data hidden in synthetic / fake public-key-like material or script patterns

# Encodings

- `OP_RETURN` and visible witness payloads are easy to identify
- Other encodings can be much less visible at policy level
- Example: data hidden in synthetic / fake public-key-like material or script patterns
- Filtering one encoding path can push usage into less transparent and more harmful ones

# Asymmetry

- Filters remove known patterns

# Asymmetry

- Filters remove known patterns
- Encoders move to new patterns

# Asymmetry

- Filters remove known patterns
- Encoders move to new patterns
- Encoders have more degrees of freedom than filters

# Asymmetry

- Filters remove known patterns
- Encoders move to new patterns
- Encoders have more degrees of freedom than filters
- This is a structural limit

# Incentives

- “Spam” definitions do not remove the limits above

# Incentives

- “Spam” definitions do not remove the limits above
- Filtering can push embedding into worse forms

# Incentives

- “Spam” definitions do not remove the limits above
- Filtering can push embedding into worse forms
- `OP_RETURN` is prunable; fake UTXO growth is worse

# Incentives

- “Spam” definitions do not remove the limits above
- Filtering can push embedding into worse forms
- `OP_RETURN` is prunable; fake UTXO growth is worse
- Incentives and harm reduction matter more than prohibition

# Miners

- Miners (more precisely pools) run nodes too

# Miners

- Miners (more precisely pools) run nodes too
- Their incentive is fee revenue and reliable block production

# Miners

- Miners (more precisely pools) run nodes too
- Their incentive is fee revenue and reliable block production
- They do not automatically share relay-policy filtering goals

# Miners

- Miners (more precisely pools) run nodes too
- Their incentive is fee revenue and reliable block production
- They do not automatically share relay-policy filtering goals
- Even strong relay filtering can fail if mining incentives remain permissive

# Rust

# Rust at Braiins

- Rust across embedded systems

# Rust at Braiins

- Rust across embedded systems
- Rust in infrastructure components

# Rust at Braiins

- Rust across embedded systems
- Rust in infrastructure components
- Rust in high-level, low-latency network services

# Async in Rust

- `async fn` lets us write non-blocking I/O code in direct style

# Async in Rust

- `async fn` lets us write non-blocking I/O code in direct style
- Futures represent work that can make progress over time

# Async in Rust

- `async fn` lets us write non-blocking I/O code in direct style
- Futures represent work that can make progress over time
- `.await` yields control while waiting for I/O

# Async in Rust

- `async fn` lets us write non-blocking I/O code in direct style
- Futures represent work that can make progress over time
- `.await` yields control while waiting for I/O
- This is a good fit when one process manages many peer connections

# Tokio

- Network software is mostly waiting: sockets, timeouts, backpressure



# Tokio

- Network software is mostly waiting: sockets, timeouts, backpressure
- Async tasks make large peer sets practical



# Tokio

- Network software is mostly waiting: sockets, timeouts, backpressure
- Async tasks make large peer sets practical
- Rust gives memory safety under concurrency



# Tokio

- Network software is mostly waiting: sockets, timeouts, backpressure
- Async tasks make large peer sets practical
- Rust gives memory safety under concurrency
- Metrics/logging compose well in the same process

```
loop {  
    tokio::select! {  
        result = stream.read(&mut buf) => { /* parse */ }  
        Some(msg) = outbound_rx.recv() => { /* send */ }  
        _ = keepalive.tick() => { /* ping */ }  
    }  
}
```

Rust

}

}

# Why Rust

- Hostile network input benefits from strong typing and explicit parsing

# Why Rust

- Hostile network input benefits from strong typing and explicit parsing
- Enums + match make protocol state handling readable and auditable

# Why Rust

- Hostile network input benefits from strong typing and explicit parsing
- Enums + match make protocol state handling readable and auditable
- Bounded channels make backpressure decisions explicit

# Why Rust

- Hostile network input benefits from strong typing and explicit parsing
- Enums + match make protocol state handling readable and auditable
- Bounded channels make backpressure decisions explicit
- You can push performance without giving up memory safety

# Patterns

- One task per peer is easy to reason about

# Patterns

- One task per peer is easy to reason about
- `tokio::select!` maps naturally to socket/read/write/keepalive loops

# Patterns

- One task per peer is easy to reason about
- `tokio::select!` maps naturally to socket/read/write/keepalive loops
- `Arc<RwLock<...>>` is enough for shared relay state and metrics in a small tool

# Patterns

- One task per peer is easy to reason about
- `tokio::select!` maps naturally to socket/read/write/keepalive loops
- `Arc<RwLock<...>>` is enough for shared relay state and metrics in a small tool
- `tracing` + `Prometheus` + `Axum` make observability cheap

# Demo Tool

- `crab-router` is a demo instrument supporting the argument

## Demo Tool

- `crab-router` is a demo instrument supporting the argument
- It connects to many mainnet peers and classifies implementations

# Demo Tool

- `crab-router` is a demo instrument supporting the argument
- It connects to many mainnet peers and classifies implementations
- It observes relay, discovery, and transaction flow metrics

# Demo Tool

- `crab-router` is a demo instrument supporting the argument
- It connects to many mainnet peers and classifies implementations
- It observes relay, discovery, and transaction flow metrics
- The point is measurement and demonstration

# **Demo**

# Demo

- Live crab-router run on mainnet

# Demo

- Live crab-router run on mainnet
- Peer mix and transaction flow in Grafana

# Demo

- Live crab-router run on mainnet
- Peer mix and transaction flow in Grafana
- Discovery traffic (addr / getaddr)

# Demo

- Live crab-router run on mainnet
- Peer mix and transaction flow in Grafana
- Discovery traffic (addr / getaddr)
- Use observations to support the topology argument

**End**

End

# Q&A