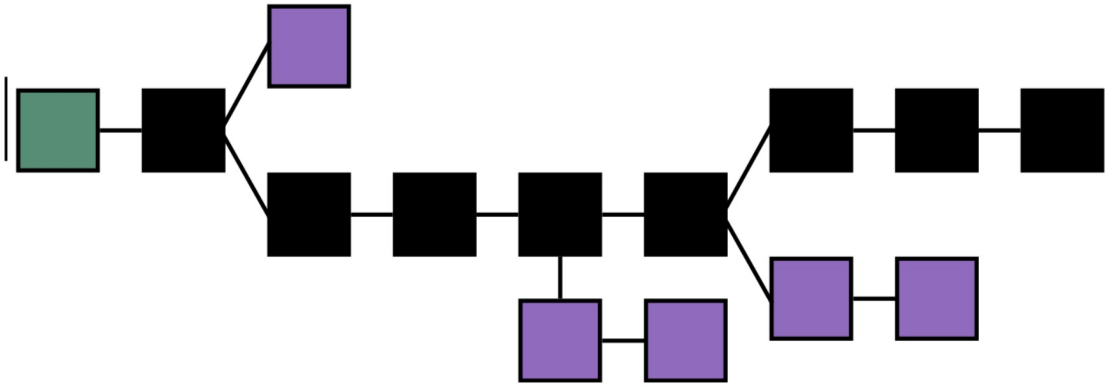
Blokų grandinių techologijos

2019/2020 ruduo

Dėstytojas Partn. Doc. Saulius Grigaitis

1 Paskaita

> A blockchain, originally block chain, is a growing list of records, called blocks, that are linked using cryptography.



- Blokuose (pažymėti kvadratais) - saugoma svarbi informacija.
- Grandinė prasideda specialiu bloku, vadinamu Genesis (pažymėta žalia spalva). Paprastai jis būna įkoduotas į klientines programas. Blokų grandinės kūrėjai į šį bloką gali įrašyti informaciją apie tam tikrą pradinį balansą savo paskyrose.
- Konfliktas kyla, jei viename aukštyje (bloko aukštis - kiekis blokų, esančių iki to bloko blokų grandinėje; *Genesis* bloko aukštis yra lygus 0) atsiranda daugiau nei vienas blokas ir abu blokai yra validūs pagal protokolą.
- Jei nebūtų taisyklės, kaip išrinkti kanoninį bloką, atsirastų problema: blokų grandinės šakos vis šakotųsi - gautume ne kanoninę šaką, o medį.

Proof of Work

- Vienas šios problemos sprendimo būdų - Proof of Work protokolas.
- Visi mazgai vienu metu bando atrasti sekantį bloką (*cryptographic puzzle* - ieškoma skaičiaus (*nonce*), kurį pridėjus prie bloko gaunamas *hash*'as, tenkinantis tam tikras savybes, pavyzdžiui prasidedantis *n* nulių). Už šio skaičiaus atspėjimą, mazgas gauna atlygį (*reward*).

> Kriptografinė maišos funkcija (*cryptographic hash function*) priskiria argumentui maišos kodą (*hash* arba *digest*). Argumentą pakeitus nors vienu bitu, *hash*'as smarkiai pasikeičia, todėl iš jo apskaičiuoti argumentą praktiškai neįmanoma. Geriausia taktika būtų bandyti visas kombinacijas iš eilės (*brute force*) ir atspėti.

- Jei atsiranda nesutarimų (grandinė išsišakoja) kanonine (teisinga) šaka laikoma ta, į kurią įdėta daugiau darbo (kitaip tariant ta, kuri yra ilgesnė).
- Jei mazgas A nuspręstų sukčiauti, pavyzdžiui, išsiųsti įrašą apie transakciją kitam mazgui B (*transakcija A perveda x valiutos B*), bet nepranešti kitiems (tam, jog šie manytų, kad pirmasis mazgas vis dar turi x valiutos), atsirastų dvi grandinės: viena su melaginga transakcija, o kita be. Tam, jog melaginga šaka būtų laikoma teisinga, A mazgas turėtų atspėti visų po to sekančių blokų *proof of work*. Tikimybė, jog tą padaryti pavyktų neturint bent apie 50% tinklo skaičiavimų, yra ypač maža.
- Tad kuo daugiau darbo įdedama į kiekvieną bloką, tuo didesnį saugumą galima pasiekti. Kuo daugiau skaičiavimo resursų tinklas turi, tuo jis yra saugesnis.

Pliusai	Minusai
Gerai *randomizuojamas* blokų *produce*'inimas.	Švaistoma labai daug energijos. Reikia daug resursų, jog tinklas būtų sąlyginai saugus. Išlieka rizika, kad kas nors turės daugiau resursų ir užvaldys tinklą. Neturi finalizavimo mechanizmo, tik produce'inimo mechanizmą.

- Neblogas video (https://youtu.be/bBC-nXj3Ng4) apie tai, kaip veikia *Bitcoin*'as.

CAP teorema

- Išskirstytose sistemose (*distributed systems*) galima turėti tik 2 iš 3:
    - Vientisumą (*consistency*). Kiekvienas skaitymas (*read*) gauna arba tą patį rezultatą, arba klaidą).
        - Aukojama dažniausiai iš visų trijų.
        - Blokų grandinėse: kiekvienas mazgas turi kanoninę šaką, tad skirtingi mazgai mato skirtingą būseną.
        - Realiame pasaulyje palaukus tam tikrą kiekį blokų, blokas yra tikimybiškai finalizuojamas - tikimybė, jog blogas bus atmestas yra labai maža.
        - *Eventually consistent* - jei tam tikri duomenys nebebus atnaujinami, galiausiai visi skaitymai (*read*) grąžins naujausią vertę. Pavyzdžiui, būnant *offline*, pavyksta pa'*post*'inti *Facebook*'e.
    - Pasiekiamumą (*availability*) - servisas visada pasiekiamas.
        - Blokų grandinėse: galima *mine*'inti nuo *Genesis* bloko.
        - Servisas, anot protokolo, veikia, tik nėra vientisumo (consistency).
    - Fragmentacijos tolerancija (*partition tolerance*) - ar toleruojama fragmentacija - pasiskirstymas į dalis.
        - Blokų grandinėse: priimama ta šaka, į kurią įdėta daugiausia darbo.

Ethereum 2.0

- Nereikalauja tiek daug resursų, kadangi nenaudojamas *Proof of Work* protokolas.
- Pasižymi labiau garantuotu blokų finalizavimu, kadangi visus, kurie daro įtaką naujų blokų gavimui, padaro dalininkais (*Proof of Stake*).
- Kad blokas būtų priimtas, už jį turi statyti validatoriai, kurie yra pastatę 32 ETH (validatoriumi tampama, užstačius 32 ETH).
    - Pavyzdys: yra 10k validatorių, kurie yra užsiregistravę ir užstatę po 32 ETH (~$6k). Visa užstatyta suma - $6M. Šia suma negalima naudotis, tačiau validatoriai gali balsuoti. Už transakciją validatorius gauna tam tikrą procentą atlygio (kadangi transakcijos kainuoja). Jei aptinkama, kad validatorius elgiasi blogai (pvz., siūlo du blokus toje pačioje epochoje) iš jo atimamas užstatas ir jis pašalinamas iš validatorių sąrašo. Pinigai paprastai sudeginami.
- Naudojamas ekonominis finalizavimas - blokai gali buti atšaukiami, tačiau kažkas tokiu atveju prarastų labai daug pinigų.
- Palaikomas trečalis piktybinių mazgų (*byzantine nodes* - mazgai, kurie elgiasi bet kaip, pavyzdžiui, balsuoja kitaip nei yra iš tikrųjų, siunčia skirtingas žinutes). Blogiausiu atveju reikia 2/3 visų balsų, jog būtų priimtas sprendimas.
- *Tendermint* protokole sustabdoma, jei nesulaukiama 2/3 balsų.

2 paskaita

Konsensusas

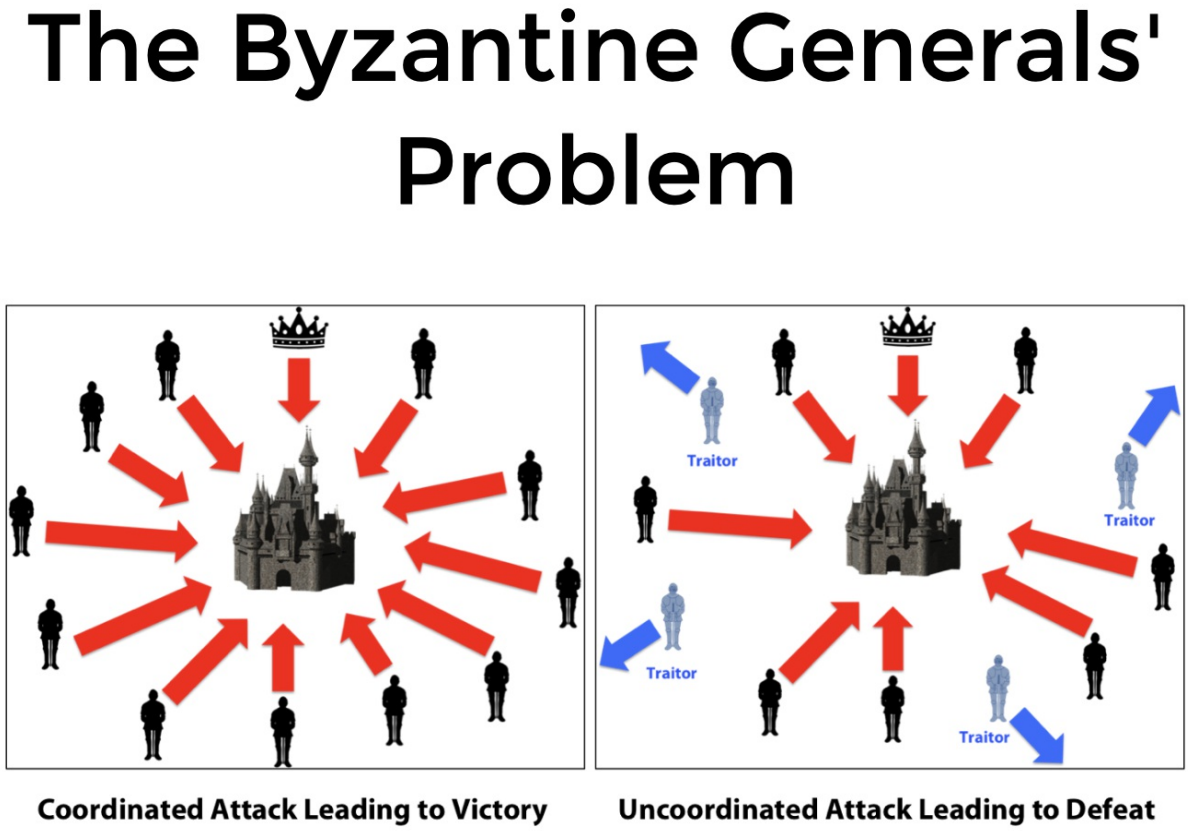
> A fundamental problem in distributed computing and multi-agent systems is to achieve overall system reliability in the presence of a number of faulty processes.

- Tinklas - grafas.
- Trikiai:
    - Nepiktavališki:
        - Pavyzdžiui, tinklo trikis, *latency*, *bitflip*.
        - Jei nebūtų šių trikdžių, būtų galima išpręsti nemažai problemų.
    - Piktavališki:
        - Piktavališki mazgai - kelia nemažai problemų.
- Trikiai (*failure modes*):
    - *Crash stop* - vienas mazgas išsijungia tokiam laikui, kad jo išsijungimą pastebėtų tinklas.
    - *Crash recovery* - mazgas išsijungia kuriam laikui. Tam tikruose konsensuso mechanizmuose atsigavęs mazgas gali būti traktuojamas kaip visiškai naujas mazgas. Pavyzdžiui, jei nedalyvavo 5 balsavimuose, mazgas panaikinamas. Nėra būdo nustatyti, ar mazgas buvo atsijungęs piktybiškai.
    - *Omission fault* - mazgas kažko iš dalies nepadaro. Pavyzdžiui, replikavimas - yra pagrindinis mazgas ir 10 mazgų, kurie pasiima kopiją ir ją išsisaugo pas save. Klaida padaroma tada, kai mazgas pasiima kopiją, ją išsisaugo, bet nepraneša apie sėkmę (kai kurios replikos *broadcast* metu gali gauti duomenis, bet nepranešti centriniam serveriui, kad gavo informaciją). Jei dalinai *apsirgusios* replikos grįžta, atsiranda keista situacija - dauguma replikų yra corrupted.

        - *R1* patvirtina, kad gavo gera informaciją, *R2* ir *R3* turi *corrupted data*. Dingsta centrinis serveris, šaukiamasi pagalbos iš replikų, žiūrima, kur daugiausia sutampa duomenys (kokius duomenis turi dauguma). Jei *R1* ir *R2* turi vienodai nekorektiškus duomenis, o *R1* - teisingus, teisingais laikomi *corrupted* duomenys.
          [C]
        / | \
        [R1] [R2] [R3]

    - *Byzantine (arbitrary)* - mazgas gali neatsakinėti, gali atsakinėti kitaip (piktybinis). Tačiau nebūtinai su bloga intencija, nes gali įvykti *bitflip*'as dėl radiacijos ir pan.

Bizantijos generolo problema



- Grupės karių puola pilį. Visi kariai (arba būriai) neturi patikimo komunikacijos kanalo. Sėkmė pasiekiama tik tada, kai *visi* nusprendžia arba pulti, arba nepulti.
- Trijų elementų pavyzdys:

Byzantine Failures: Commander Traitor

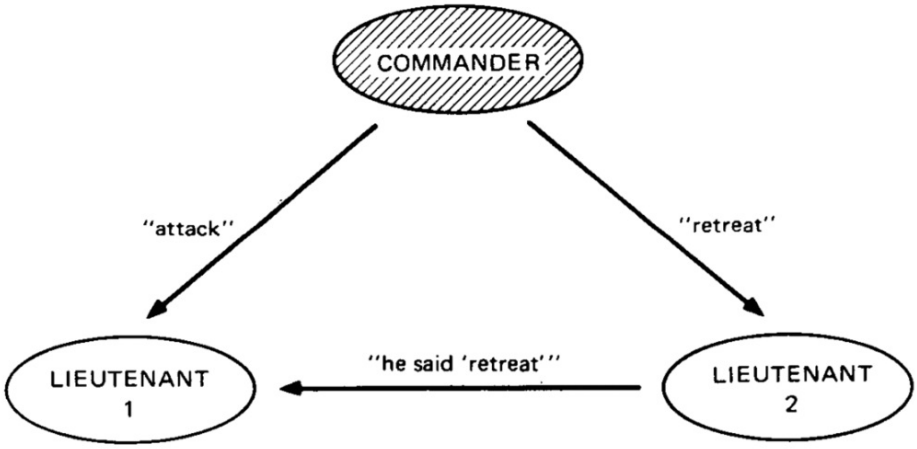


Fig. 2. The commander a traitor.

Byzantine Failures: Lieutenant Traitor

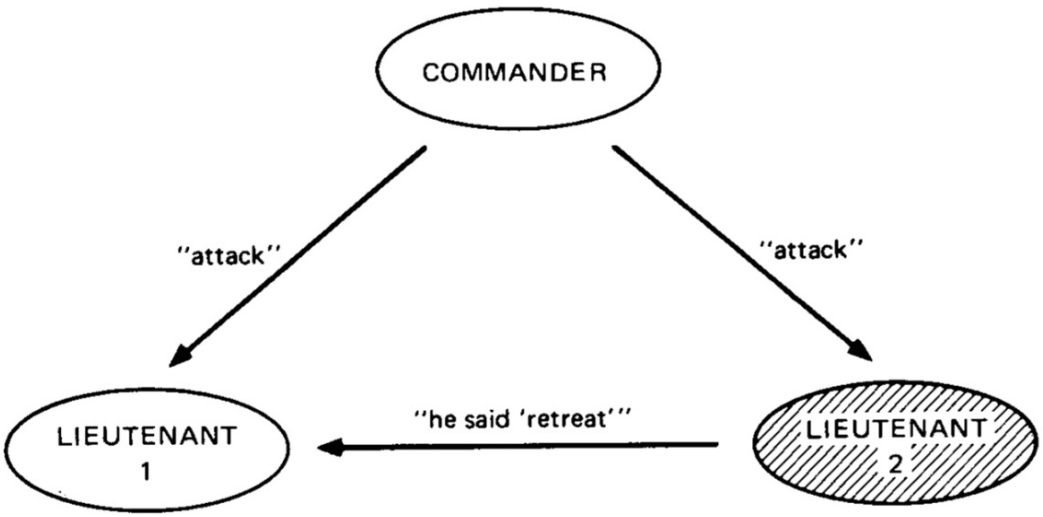


Fig. 1. Lieutenant 2 a traitor.

No Solution With Fewer Than 3m+1 Generals for m Traitors.

- Jei kanalas nepatikimas - nėra sprendimo, su mažiau nei *3m+1* mazgu, kai blogiečių yra *m*.
- *Byzantine faults* nutinka dažnai. Jei žinutės nepasirašytos, reikia 4 atskirų mazgų, kurie atpažintų vieną blogą.
- Apsisaugojimui reikia daug perteklinės įrangos. Pavyzdžiui, kosminiuose aparatuose yra trys kompiuteriai, kurie skaičiuoja tą patį.

FLP imposiibility (1985)

> In a fully asynchronous message-passing distributed system, in which one process may have a crash failure it is not possible to solve the consensus problem.

- Pilnai asinchroninis tinklas su bet kiek mazgų negali pasiekti konsensuso bent vienam mazgui patyrus *crash fault*.
- FLP: galima turėti tik du iš trijų:
    - *Safety* - the nodes agree on a valid value that was proposed by one of the nodes. *Real life example*: antras prezidento rinkimų turas, o išrenkam S. Skvernelį :)
    - *Liveness* - the nodes eventually reach agreement. *Real life example*: pažeidimas per referendumą dėl dvigubos pilietybės - nubalsavome, bet niekas nenuspręsta.
    - *Fault tolerance* - the system can survive a failure.

Timing modes

- Synchronous model (retai sutinkamas dėl tinklų).
- Asynchronous model (FLP įrodytas tokiame tinkle: nėra laiko apribojimo žinutės siuntimui arba jos vykdymui).
- Semi-synchronous model (žinutės ne iš karto pasiekia adresatą ir yra įvykdomos, bet yra laiko apribojimai (*timeout*)).

Teoriniai įrodymai

> Jei nėra teorinio modelio, dalykas gali veikti, bet gali ir neveikti.

Teorinis įrodymas	Eksperimentinis įrodymas
Įrodyta ir viskas.	Galima kažką sugalvoti ir įrodyti. Kol niekas negali eksperimentu to paneigti, mokslas priima tokią teoriją kaip teisingą.

- Eksperimentai turi būti *repeatable* ir *reproduceable*.

Whitepapers vs *scientific papers*

- *ICO* bumas: whitepapers - marketinginė dalis. Naudojamas dirbtinis formalizavimas, mokslinė kalba.
- *Whitepaper*'iuose dažniausiai nebūna eksperimentų, *peer review*'ų.
- Kartais galima rasti dalinius įrodymus.

3 paskaita

Praktinė dalis

- *Ethereum 2.0 specification*:
    - *BLS signatures - implementation in ethereum differs from official specification.*
    - *Details might be missing from specification.*
    - *Implementations:*
        - *Shasper (by Parity);*
        - *Lighthouse.*
    - *YAML tests.*
    - *unit tests (RUST language difficulties in unit tests).*

MetaMask - blockchain cannot be accessed via HTTP protocol. MetaMask allows interaction with blockchain (eg - transfer tokens) via browser add-on.

Wallets

- Blokų grandinės yra apsaugotos kriptografinais raktais.
- Kriptografiniai raktai negali būti perkurti (sukurti iš naujo, *recreated*).
- Sudaro du raktai: privatus (*secret key*) ir viešas (*public key*). Informacija pasirašoma (*sign*) privačiu raktu ir gaunamas parašas. Parašas gali būti verifikuojamas (*verify*) viešai prieinamu viešu raktu.
- Yra du *wallet* tipai:
    - Kažkas kitas saugo privatų raktą (prisijungimui (prie puslapio) naudojamas el. paštas ir slaptažodis. Saugumo spraga: kas nutiks, jei raktai bus pavogti.
    - Vartotojas pats saugo raktą:
        - Faile (apsaugotame *passphrase*).
        - *Brain wallets (12 words that generate unique key).*

4 paskaita

- Pristatymai:
    - Transition function (https://docs.google.com/presentation/d/19OTFduVJMW1gwOX1kVEP0JXkJbjJ1ivW7H0Pl usp=sharing)
    - SSZ & Merkle Proof (https://docs.google.com/presentation/d/1wd00AF0U6iSKgwlj86ZgOzNvhiEM-19bWZj8mQn9c1s/edit?fbclid=IwAR02DQ-FP6SoLsh3FAteXxiJP8ST-zDlwi3LWNdI14XQzMIsst8fxkra6V8)
    - Helper functions (https://docs.google.com/presentation/d/11jgzgwLIRGQ3s0RCnMngXSH7J7o9oWbxFn1g3 usp=sharing)

5 paskaita

- Pristatymai.

Praktinė dalis (tęsinys)

- *Lint automatically runs by running scripts/ci.*

- *Use cargo/check before ci script.*

6 paskaita

Rust programavimo kalba

> kokia programavimo kalba jei nėra kintamųjų - HTML :-)

assignment

```
let x = 5;
```

- in case a new value might be assigned use mut before variables;
- const can't be mut;
- const can be used at global scope;
- *shadowing* - redeclaration of variables is possible.

types

```
let x: u32 = 5;
```

- arch - depends on system (32/64 bits): isize, usize;
- number literals:
    - 98_222 - decimal;

```
let x = 5.00; // f64
```

- bool;
- char (UTF-8 supported);
- tuple - a set of variables (might be separate types);
- array ([1, 2, 4];) - length must be known at compile time;

functions

- fn main() - program entry point;

```
fn function(x: 32) -> i32 {
  // do something
  5
}
```

- last sentence without ';' and return statement is the result of expression;

control flow

- if <condition>
- if <condition> <...> else <condition>

```
let number = if condition {
  7
} else {
  6
};
```

- loop;
- while;
- for element in a.iter { };

ownership

- Some languages give freedom and responsibility to programmers to allocate / free memory. Involves high risk.
- Others use GC - garbage collector (drawbacks include program *freezing*, unrecognized references etc);
- Rust uses a different approach:
    - Checks memory management at compile time (*vs runtime*);
    - Can't have two references to same object;
    - & marks reference;
    - mut& allows edits of referenced object;
    - slice &[0..5] allows to return part of referenced value;

structures

```
struct User {
  name: String
  username: String
  active: bool
}

let user = User {
  email: email,
  username: username,
  active: true
}

struct Color(i32, i32, i32)

impl User {
  fn activate(&self) { // method
    self.active = true
  }
}
```

- multiple impl are available;

enums

```
enum Coin {
  Penny,
  Nickel,
  Dime,
  Quarter
}

match coin {
  Coin::Penny => 1
  Coin::Nickel => 5
  Coin::Dime => 10
  Coin::Quarter => 25
}
```

option

- generic type transfering another type;
- None;
- Some();

packages, crates, modules & paths

- new package - cargo new my-project;
- mod - namespaces;
- file names work as mod;
- pub makes function public (can be *imported*);
- use crate::front_of_house::hosting;
- as IoResult - alias;
- pub use -> re-export;

```
[dependencies]
rand = "0.5.5"
```

- caution -> different crates can use different versions. Types may interfere and show cryptic messages;
- * - import all;

generics & traits

- generic function:

```
fn largest<T: PartialOrd>(list: &[T]) -> T { // trait bound
  let mut largest = list[0];

  for &item in list.iter() {
    if item > largest {
      largest = item;
    }
  }

  largest
}
```

- generic struct:

```
struct Point<T, U> {
  x: T,
  y: U,
}
```

- generic enum:

```
enum Result<T, E> {
  Ok(T),
  Err(E),
}
```

```
impl<T> Point<T> {
  fn x(&self) -> &T {
    &self.x
  }
}
```

- traits:

```
pub fn notify(item: impl Summary) {
    println!("Breaking news! {}", item.summarize());
}
```

- multiple traits are separated by '+';

7 paskaita

Rust programavimo kalba (tęsinys)

vector

- macro to create:

```
let v = vec![1,2,3]
```

- variable size;
- access with get;
- can wrap types in enums to store different types in a single vector;

  subtle difficulties with UTF8 strings.

hashmap

- insert with insert;
- iterate:

```
for (key, value) in &scores { }
```

- you can override values;

- or_insert adds only when key does not exist;

error handling

- unrecoverable with panic!;
- stack trace with RUST_BACKTRACE=1;
- recoverable errors with Result;

```
enum Result<T, E> {
   Ok(T),
   Err(E),
}
```

- pattern matching;
- unwrap_or_else;
- propagate errors by returning Result and wrapping Error;
- ? - auto propagates error;
- error can be ignored when we know it won't happen (parsing hardcoded string);
- create / access logic with private struct field and public getter;

lifetime

- lifetime 'a or 'b (can be any letter);
- usually inferred;
- might be needed when passing pointers;
- static lifetime (bad practice, only for things like config);

tests

```
#[cfg(test)]
 mod tests {
   #[test]
   fn it_works(){
     assert_eq!() // todo
   }
 }
```

- assert!(x) checks if x is true, optional message param;
- assert_eq!(x, y) checks equality;
- panic breaks test suit, avoid with #[should_panic] - only passes if test panics;
- error handling as usual;

iterators and closures

- closure:

```
let expensive_clojure = |num| {
  println!()...
}
```

- doesn't require types

- iterators impl trait with method next;

- after adapters (like map) producer (like collect) needs to be called;

smart pointers

- Box<T> allows self referencing structures;
- boxed values are kept in heap;
- vec<Box<dyn Draw>> allows storing obejcts that implement trait;
- loses some performance;

9 paskaita

Praktinė dalis (2 tęsinys)

- lint, lint, lint
- important requirement for next task => prove that module is working correctly
- don't use asserts and don't panic

10 paskaita

Concurrent programming with *Rust*

- memory safety guaranteed;
- need to join threads, otherwise main stops executing and other threads are killed;
- assign handle and handle.join();
- if variables are used move keyword is required;

> do not communicate by sharing memory, share memory by communicating.

- mpsc::channel();
- tx - transmitter;
- rx - receiver;
- recv() is blocking;
- sent vals can't be used;
- rx treated as iterator;
- clone transmitter to use from separate threads;
- *green threads* - created by runtime, portable, only one thread;
- *OS threads* - not portable, but can support multiple threads;
- Rust has minimalistic runtime (OS thread 1:1 green thread);

```
let m = Mutex::new(5)
let mut num = m.lock().unwrap();
*num = 6;
```

- Arc<T> required to pass lock around threads;
- *num += 1 is not atomicm so Arc is required;
- Send trait
- Sync trait

Multiple threads	Epoll (async/await)
overhead (*context switching* - resource heavy)	*event loop* - more complicated than threads.

easier to program	hard to program used when multiple clients required

- async / await in Rust;
- good for networking;

12 paskaita

- *coins* (turi savo blokų grandinę, pavyzdžiui, *Ether* funkcionuoja *Ethereum* blokų grandinėje).
- *tokens* (naudojasi egzistuojančiomis blokų grandinėmis, pavyzdžiui *Tether* naudojasi *Ethereum* blokų grandine).
- *Ether*: naudojamas transakcijoms / vertės pernešimui;
- *Tether*: pakankamai stabilus, kaina apie $1, simuliuoja *stable coin*. Didžiąją dalį transakcijų pagrindinėje *Ethereum* grandinėje sudaro būtent *Tether* transakcijos. *Tether* buvo perkeltas į *Ethereum* blokų grandinę iš *omni*.
- *Libra* (Facebook project): nebelieka *crypto-anarchy*, atsiranda reguliacijos, centralizacija.

Kontraktai realiame gyvenime

- language to specify terms of the agreement;
- a way to specify your identity and consent;
- enforcement and dispute execution;

smart contracts

> *smart contract* - a user-defined programs running on top of a blockchain. Conceptualized by Szabo in 1994.

- *smart contracts* simulates trusted third party with shared state;
- savybės:
    - Guaranteed to execute correctly;
        - Malicious miner cannot cheat;
    - Transactions are all-or-nothing;
    - Autonomous: Enforced by network;
        - Cannot be changed or stopped, even by its creator;
    - All data is stored on the blockchain;
        - Auditable history
- *smart contract* problema:
    - klaidos (*bugs*) turėtų būti ignoruojami (*code is law*);
    - klaidos (*bugs*) turėtų būti taisomos;
- *DAO hack* ir *hard fork* - kodas buvo ignoruotas (dau milijonų dolerių buvo *unfreezed*);

> Mt. Gox was a bitcoin exchange. Filed for bankruptcy after 850,000 bitcoins belonging to customers and the company were stolen.

Panaudojimas (*use cases*)

- žetonai (*tokens*):
    - *contract* stores everyone's balance;
    - *transfer* moves tokens from one account to another;
- loterijos (*lotteries*):
    - *contract* stores the end time, ticket cost, current pool, and pot;
    - If the party has enough money, add them to the pool and their money to the pot;
    - At the end, select a winner;
- draudimas;
- *cryptocurrency exchanges*;

Traditional contracts vs. smart contracts

	Traditional	Smart
Specification	Natural language + “legalese”	Code
Identity & consent	Signatures	Digital signatures
Dispute resolution	Judges, arbitrators	Decentralized platform
Nullification	By judges	????
Payment	As specified	built-in
Escrow	Trusted third party	built-in

Apribojimai

- *Smart contracts* negali būti naudojami viskam, nėra jungties su realiu pasauliu.
- Tai galima spręsti naudojant orakulus *(oracle - a trusted third party that retrieves and verifies external data for blockchains and smart contracts through methods such as web APIs or market data feeds)*.

> Ligtning network is a decentralized network using smart contract functionality in the blockchain to enable instant payments across a network of participants.

Šaltiniai

- Saulius Grigaitis, Partn. Doc. The Science Behind Blockchains (http://slides.com/saulius/byzantine-fault-tolerance-in-blockchains-6-9)
- Raymond Cheng, Dawn Song. Smart Contracts (https://berkeley-blockchain.github.io/cs294-144-s18/assets/docs/02-smartcontracts-jan-29-2018-v2.pdf)