FS19

DISTRIBUTED APPLICATIONS

BitTorrent, Tor, Rsync, Message Queues, and others

T. Bocek

thomas.bocek@hsr.ch

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Agenda

- **Distributed Systems in the News**
- **Big Picture, CAP**
- **Rsync, Tor, BitTorrent**
- **Message Queues / RabbitMQ**

Distributed Systems - In the *News*

- **29.4.2019: Peer-to-Peer video delivery explained: the story of distributed networks from the early days of the internet to the Zettabyte era**
  - Use P2P in addition to CDN is becoming popular (attention, this is their product!)
  - P2P History: UUCP, IRC, Napster
  - BitTorrent / DHT
  - “biggest limitations, viewers were required to install an additional browser plugin, which was often perceived as intrusive”
  - WebRTC: a new age in peer-to-peer video delivery
    - Supported by Chrome, Firefox, Opera, and Safari, soon Edge

- **April 2019: Local-first software - You own your data, in spite of the cloud**
  - Google Docs to collaborate, communicate with colleagues using Slack, track tasks in Trello, … taking notes, planning projects or events, remembering contacts
  - “the cloud gives us collaboration, but old-fashioned apps give us ownership”
  - Can we have both?
  - Compare current approaches

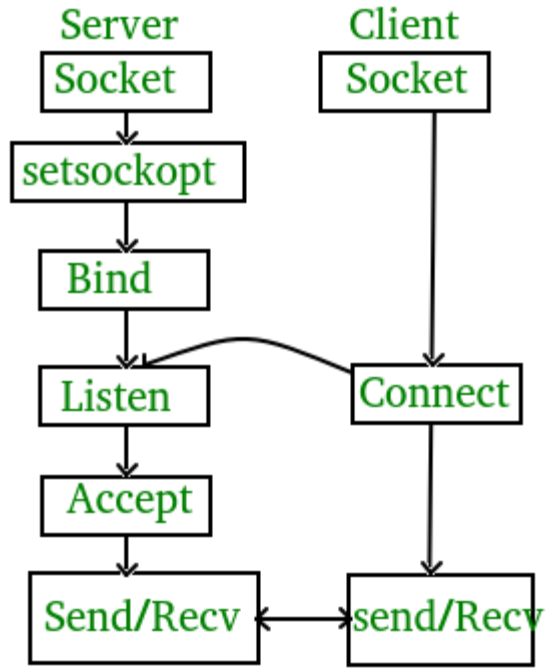
Dropbox, Google Drive, Box, OneDrive, etc.

	1. Fast	2. Multi-device	3. Offline	4. Collaboration	5. Longevity	6. Privacy	7. User control
Dropbox	✓	—	—	✗	✓	—	✓

  - Your own Nextcloud + FolderSync installation? Collabora/Onlyoffice

From the exercises

- **Socket Programming in C**
  - OS abstraction for sending and receiving data
  - Endpoint of a network protocol stack (W07-slide7)
  - Protocol stack provided by OS (exceptions, e.g., QUIC W07-slide18)
- **Server**
  - 1. create socket, set comm. type (TCP, UDP), set options
  - 2. bind to address (e.g., IP, port)
  - 3. listen – socket in listening mode
  - 4. accept – gets first connection requests, returns connected socket
- **Client**
  - 1. create socket
  - 2. connect to address (e.g., IP, port)



From the exercises: c vs. golang

```c
int main() {
    int socket_desc, client_sock, addr_size, read_size;
    char buffer[16];
    struct sockaddr_in server, client;

    //Create socket
    socket_desc = socket(AF_INET, SOCK_STREAM, 0);

    server.sin_family = AF_INET;
    server.sin_port = 7000;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");

    //Bind
    bind(socket_desc, (struct sockaddr *) &server, sizeof(server));
    listen(socket_desc, 1);

    //Accept
    addr_size = sizeof(struct sockaddr_in);
    client_sock = accept(socket_desc, (struct sockaddr *) &client, &addr_size);

    //Receive
    read_size = recv(client_sock, buffer, 16 , 0);
    printf("%d, %s", buffer[0], buffer + 1);
    //Cleanup and close... todo
    return 0;
}
```

```go
func main() {
    buffer := make([]byte, 15)

    //Create socket and bind
    tcpConn, _ := net.Listen("tcp", ":7000")

    //Accept

    conn, _ := tcpConn.Accept() //do this in a go routine

    //Receive
    n, _ := conn.Read(buffer)
    fmt.Printf("connecting... read: %d, addr: %v, data: [% x],
        decoded: %v\n", n, conn.RemoteAddr(), b[:n], string(b[1:]))
}
```

From the exercises

```c
int main() {
    int sock;
    char buffer[16];
    struct sockaddr_in server;

    //Create socket
    sock = socket(AF_INET, SOCK_STREAM, 0);

    server.sin_family = AF_INET;
    server.sin_port = 7000;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");

    //Connect
    connect(sock, (struct sockaddr *) &server, sizeof(server));

    //Send
    buffer[0] = 5;
    strcpy(buffer + 1,"Anybody there?");
    send(sock, buffer, 16, 0);

    //Cleanup and close... todo
    return 0;
}
```

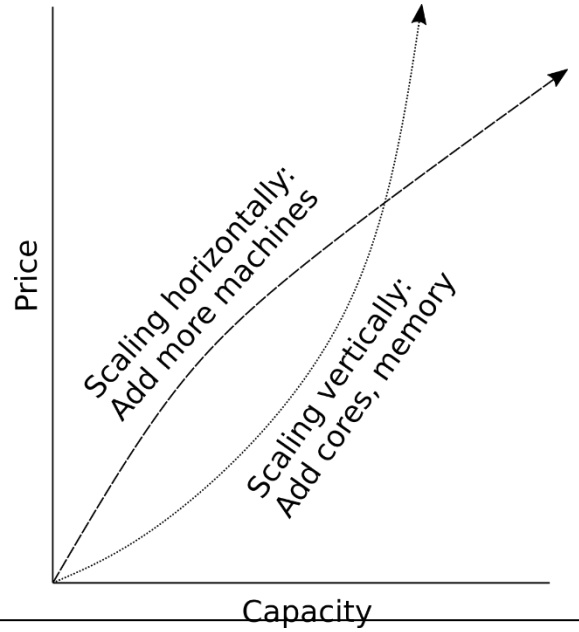
```go
func main() {
    buffer := make([]byte, 15)

    //Create socket and connect
    conn, _ := net.Dial("tcp", "127.0.0.1:7000")
    defer conn.Close()

    //Send
    buffer[0] = 5
    copy(buffer[1:], []byte("Anybody there?"))
    n, _ := conn.Write(buffer)
}
```

URL: b.socrative.com/student/

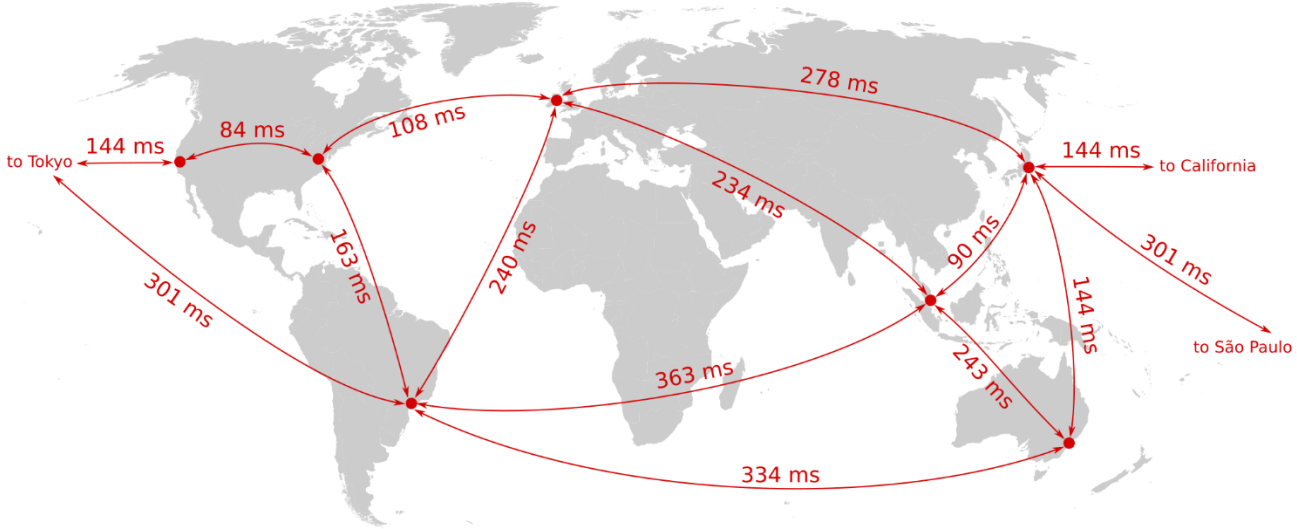
Room: HSR

Distributed Systems Motivation

- **Why Distributed Systems**
  - Single system can be fast – W8-slide12
  - Scaling: horizontally (distributed systems), or vertically – add HW
    - Apple has 75'000 Apache Cassandra nodes storing 10 petabytes of data in 2015



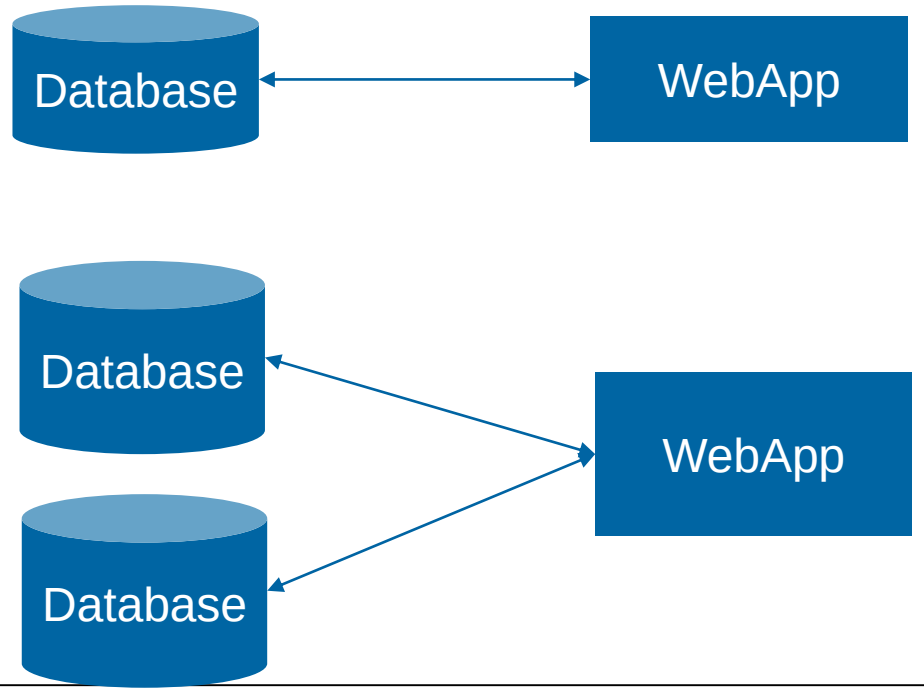
- **Performance**
  - Moore's Law – nr. of transistors doubles every 18 month
  - Nielsen's Law: a high-end user's connection speed grows by 50% per year
  - Kryder's Law: disk density doubling every 13 month (SSD, cloud) – still valid?
- **Everything gets faster, latency stays**
  - Physically bounded by the speed of light



https://www.inkandswitch.com/local-first.html

Distributed Systems Categorization

- **It is useful to classify distributed systems as either tightly coupled, meaning that the processing elements, or nodes, have access to a common memory, and loosely coupled, meaning that they do not [reference]**
  - In this lecture, distributed systems ≈ loosely coupled
- **A homogeneous system is one in which all processors are of the same type; a heterogeneous system contains processors of different types**
  - In this lecture, distributed systems ≈ heterogeneous system
- **Small-scale system: WebApp + database vs. large-scale with more than 2 machines**
  - In this lecture, often distributed systems ≈ large-scale system
- **Decentralized vs. distributed**
  - Decentralized == distributed in the technical sense, but not owned by one actor



Distributed Systems Categorization

- **Another classification**
- **CAP theorem - states that a distributed data store cannot simultaneously be consistent, available and partition tolerant**
  - Consistency—Every node has the same consistent state
  - Availability— Every non-failing node always returns a response
  - Partition Tolerant—The system continues to be consistent even when network partitions (W10-slide10)
- **With network partition - choose between consistency and availability**
  - Is a system AP or CP?
- **Blockchain and CAP**
  - Both, if you wait for n blocks, CP, if you don't AP
- **Cassandra AP**
  - But can be configured CP

Distributed Systems Categorization in this Lecture

“Controlled” Distributed Systems

- Low churn (W10-slide10)
- Examples:
  - Amazon DynamoDB (Prof. Mehta)
  - WebRTC (browser to browser communication) (~hybrid - distributed systems advanced)
  - Client/server (W10-slide21)
- “Secure environment”
- High availability
- Can be homogeneous / heterogeneous
- Mechanisms that work well:
  - Paxos, Raft (W10-slide7)
  - Can also run CRDT

“Fully” Decentralized Systems

- High churn (W10-slide10), NEO exercise
- Examples:
  - BitTorrent (this lecture)
  - WebRTC (browser to browser communication) (~hybrid – distributed systems advanced)
  - Blockchain (Prof. Mehta and upcoming lecture)
- “Hostile environment”
- Unpredictable availability
- Is heterogeneous
- Mechanisms that work well:
  - CRDT (W9-slide38)
  - Can also run Paxos (W10-slide10)

Distributed Systems Categorization in this Lecture

“Controlled” Distributed Systems

- Mechanisms that work well:
  - Consistent hashing (DynamoDB, Cassandra)
  - Master nodes, central coordinator (W8-slide11)
- Need protocols TCP/UDP
  - RPC (gRPC, custom) – W10-slide13+
- Latency matters, Google uses internally gRPC
  - Corporations tend to use standards
    LinkedIn’s Kafka – own protocol
- Network is under control or client/server → no NAT issues (WebRTC!)

“Fully” Decentralized Systems

- Mechanisms that work well:
  - Consistent hashing (DHTs) W8-slide17
  - Flooding/broadcasting - Bitcoin (Prof. Mehta and upcoming lecture) W8-slide13
- Need protocols TCP/UDP
  - RPC (gRPC, custom) – W10-slide13+
- Often custom protocols, BitTorrent (µTP)
  - If it’s a hobby project, write your own protocol, its much more fun! Latency not that critical (depends on application)
- NAT and direct connectivity huge problem (W7-execises)

Distributed Systems Categorization in this Lecture

“Controlled” Distributed Systems

- Transparency principles apply - W8-slide7
- Consistency
  - Leader election (Zookeeper, Paxos, Raft)
- Replication principles
  - Higher availability, higher reliability, higher performance, better scalability, but: requires maintaining consistency in replicas

“Fully” Decentralized Systems

- Transparency principles apply - W8-slide7
- Consistency
  - Weak consistency: DHTs
  - Nakamoto consensus (aka proof of work, only in fully distributed systems)
  - Proof of stake – Leader election, PBFT protocols, (distributed systems advanced)
  - Is Bitcoin eventually consistent?
    - Some argue no, some argue it has even stronger guarantees
    - “guarantees serializability, with a probability that is exponentially decreasing with latency”
- Replication in DHTs W9-slide38, principles apply to controlled distributed systems as well

Distributed Systems Categorization in this Lecture

“Controlled” Distributed Systems

- Mechanisms that work well:
  - BitTorrent-style data distribution W9-slide3
  - Bloomfilters reduce data transfer W9-slide15
    - Also used in other areas: DB query optimization
      - data stored in a database can be inserted into a probabilistic filter. Later, prior to querying for the data, the filter can be consulted to test whether the data exists. When the filter response is negative, an unnecessary DB query can be avoided.
  - Rsync-like mechanisms (this lecture)
- Mechanisms for Key-Value DBs (list), DynamoDB, Redis, MemcacheDB
  - Similarity search with Fast Similarity Search, Range queries with DST (W9-slide23, W9-slide28), caching, versions

“Fully” Decentralized Systems

- Mechanisms that work well:
  - BitTorrent (this lecture)
  - Bloomfilters reduce data transfer over network W9-slide15
  - Rsync-like mechanisms (this lecture)
  - Merkle-proof/hash, hash tree (this lecture)
- Key-Value Mechanisms can be used in DHTs
  - Similarity search with Fast Similarity Search, Range queries with DST (W9-slide23, W9-slide28), caching, versions

URL: b.socrative.com/student/

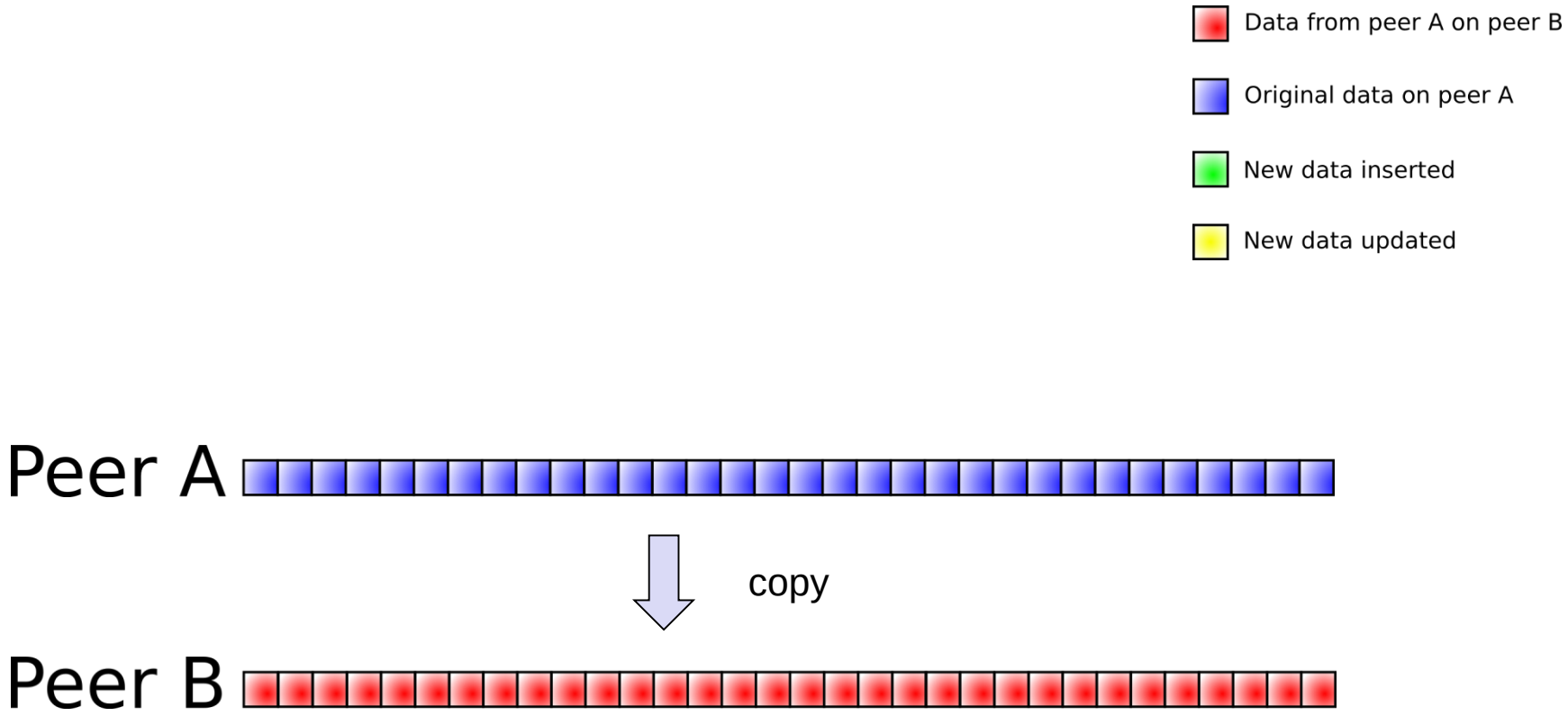
Room: HSR

Rsync - Introduction

- **Rsync used to synchronize data over network**
  - Minimizing data transfer (delta)
- **Command line client (standard utility)**
  - E.g. `rsync -aP --link-dest=$HOME/Backups/current /path/to/important_files $HOME/Backups/back-$date`
  - Unchanged files are hard linked (`--link-dest`) → Can be used for incremental backups
- **Main idea**
  - Receiver compute two checksums (strong, weak) → sent to sender
  - Sender computes with weak checksum and checks for known blocks
  - Sender verifies with strong checksum → sends difference to receiver
- **Example with two peers:**

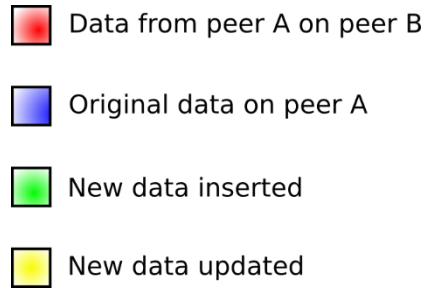
Rsync - Example

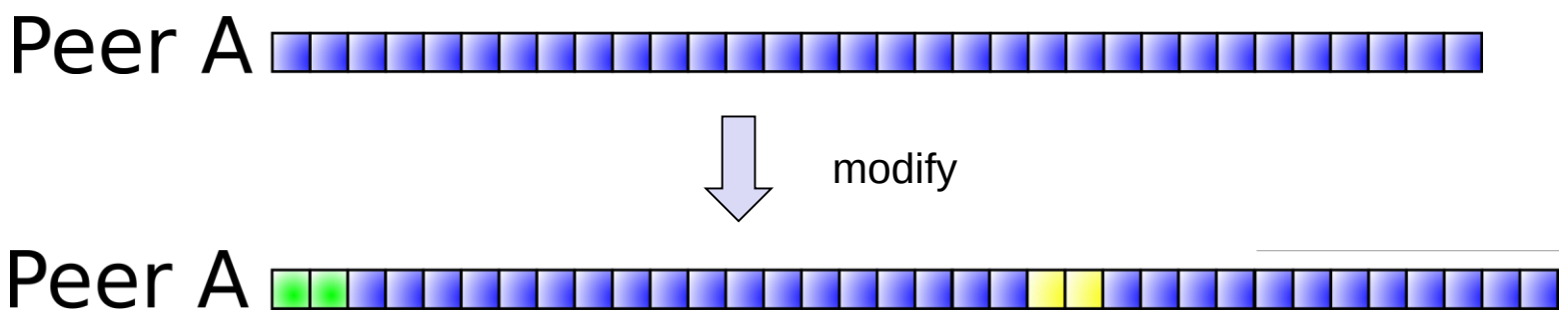
- **Peer B does not have the data → peer A copies it to peer B, no need for rsync**



Rsync - Example

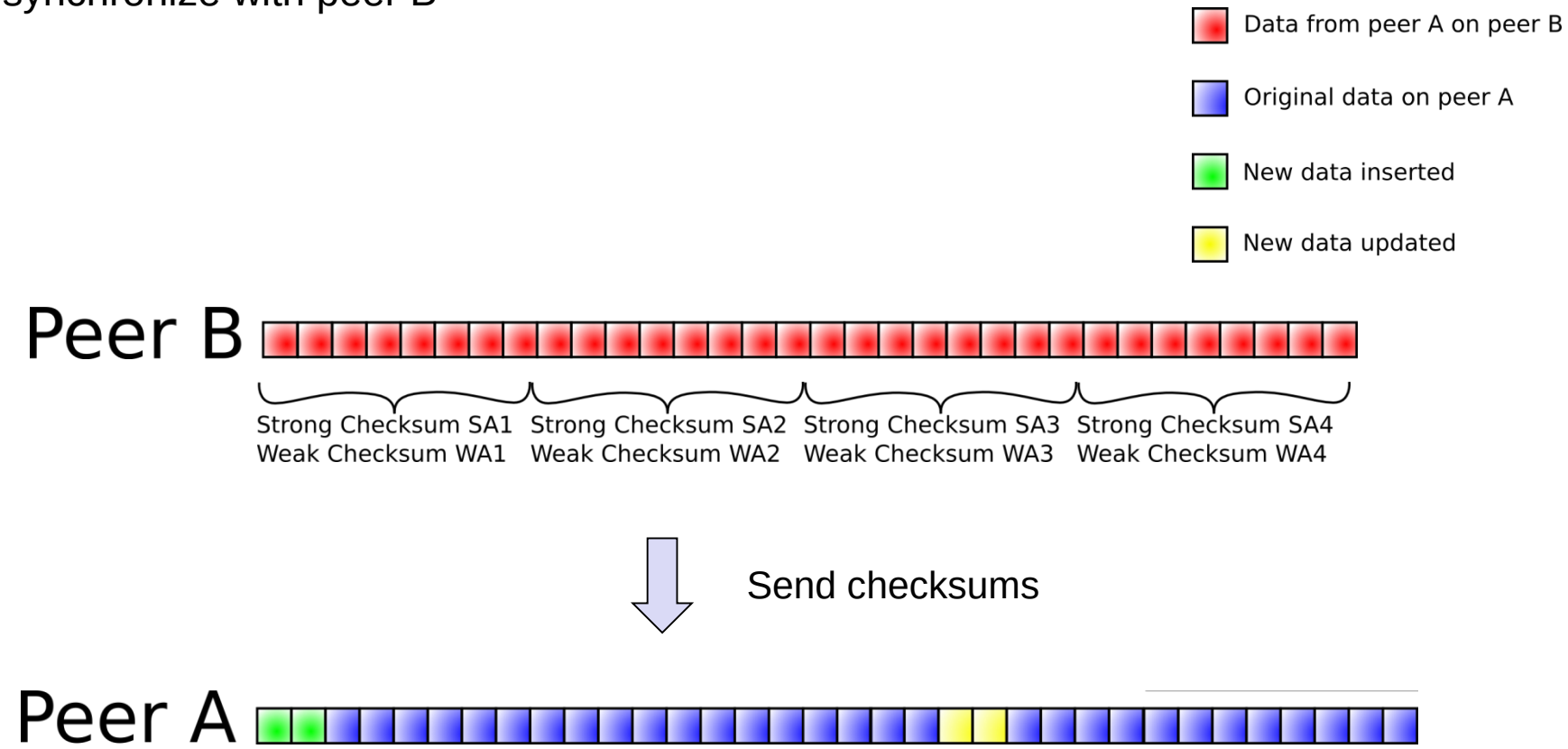
- **Peer A modifies data (insert, update)**
  - Wants to synchronize with peer B



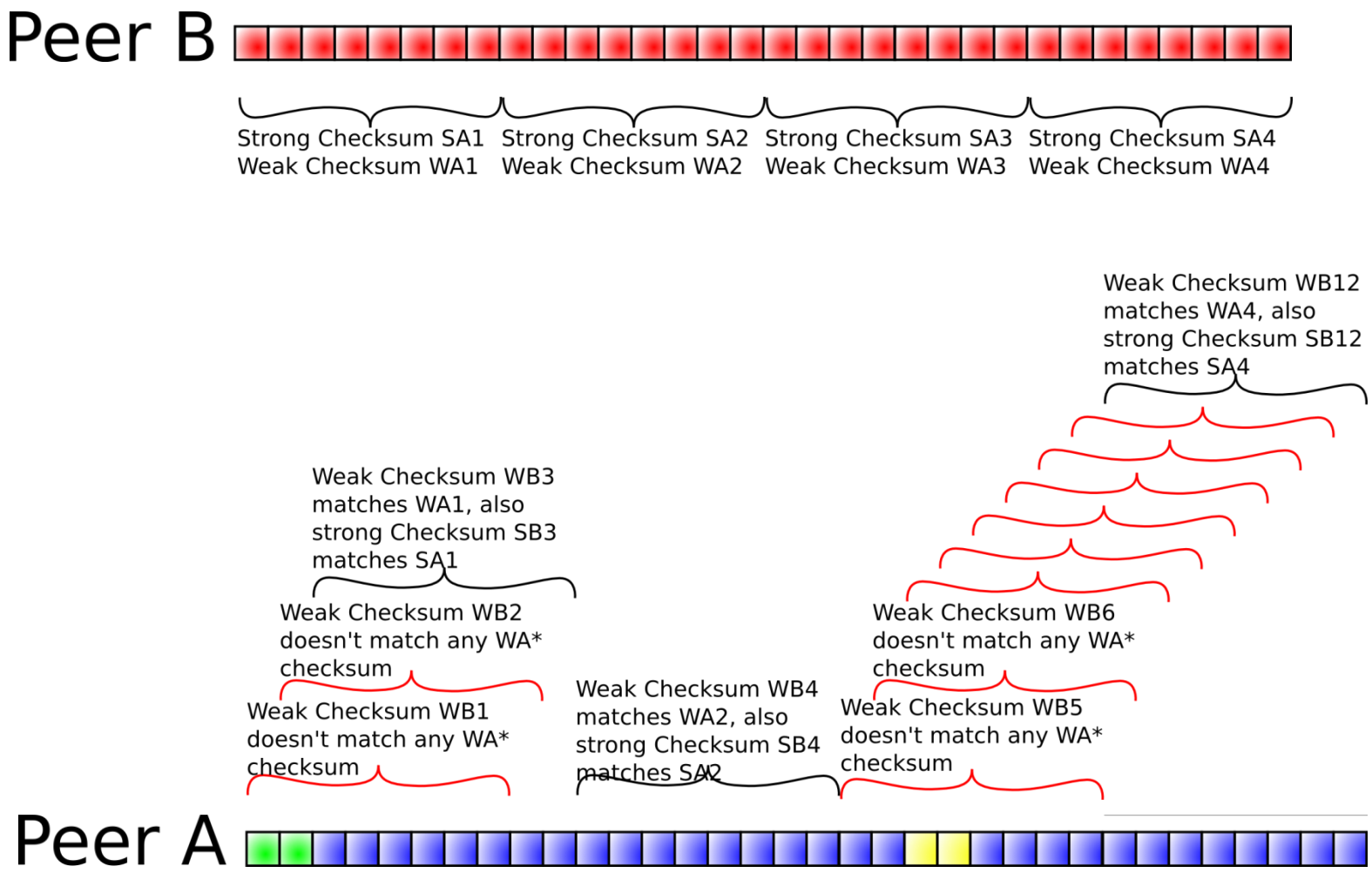


Rsync - Example

- **Peer A modifies data (insert, update)**
  - Wants to synchronize with peer B

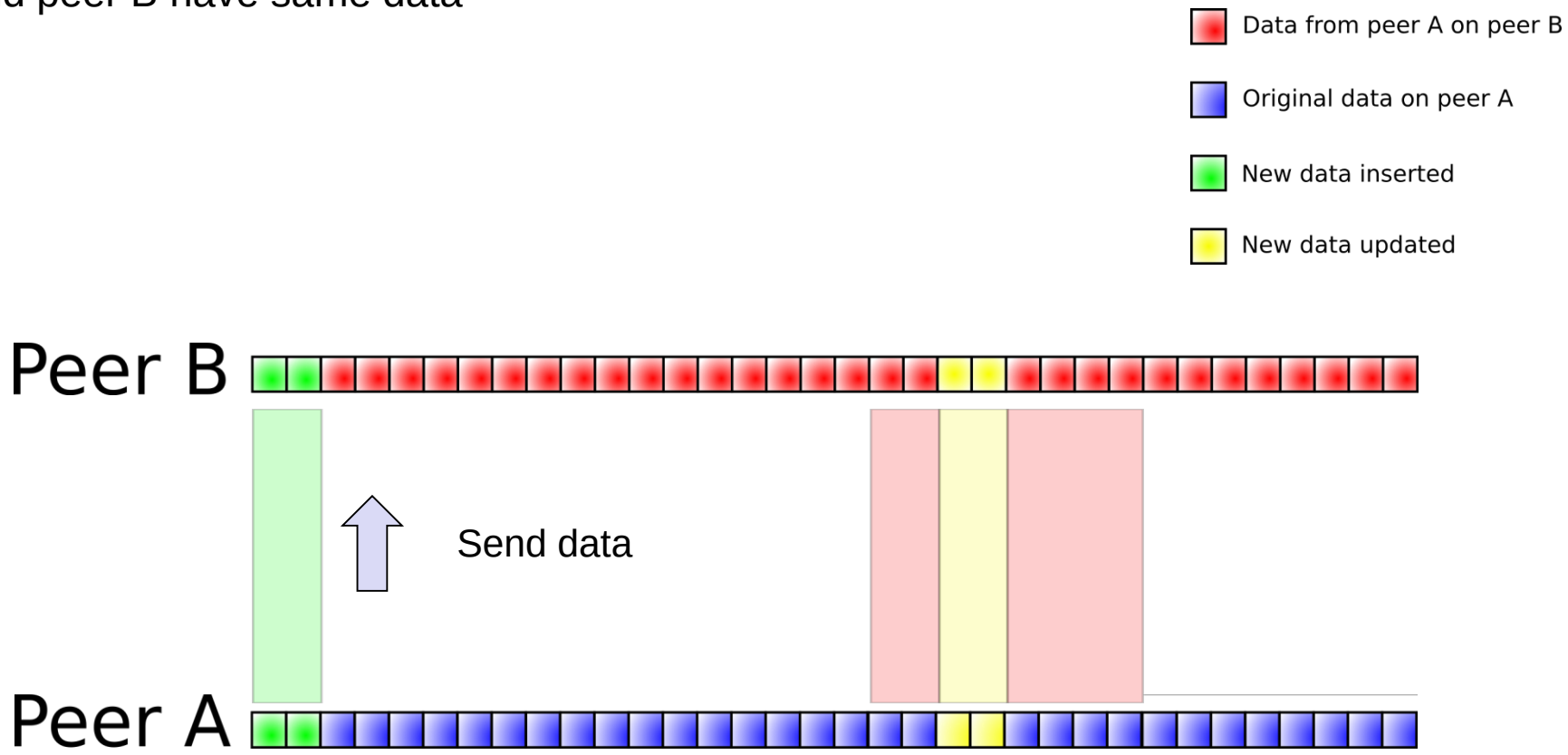


Rsync - Example



Rsync - Example

- **Peer A sends 2 + 8 blocks to peer B**
  - Peer A and peer B have same data

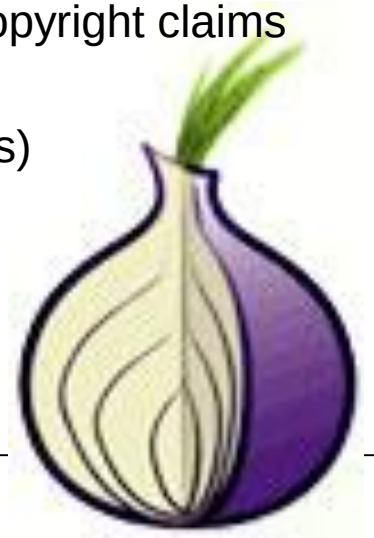


URL: b.socrative.com/student/

Room: HSR

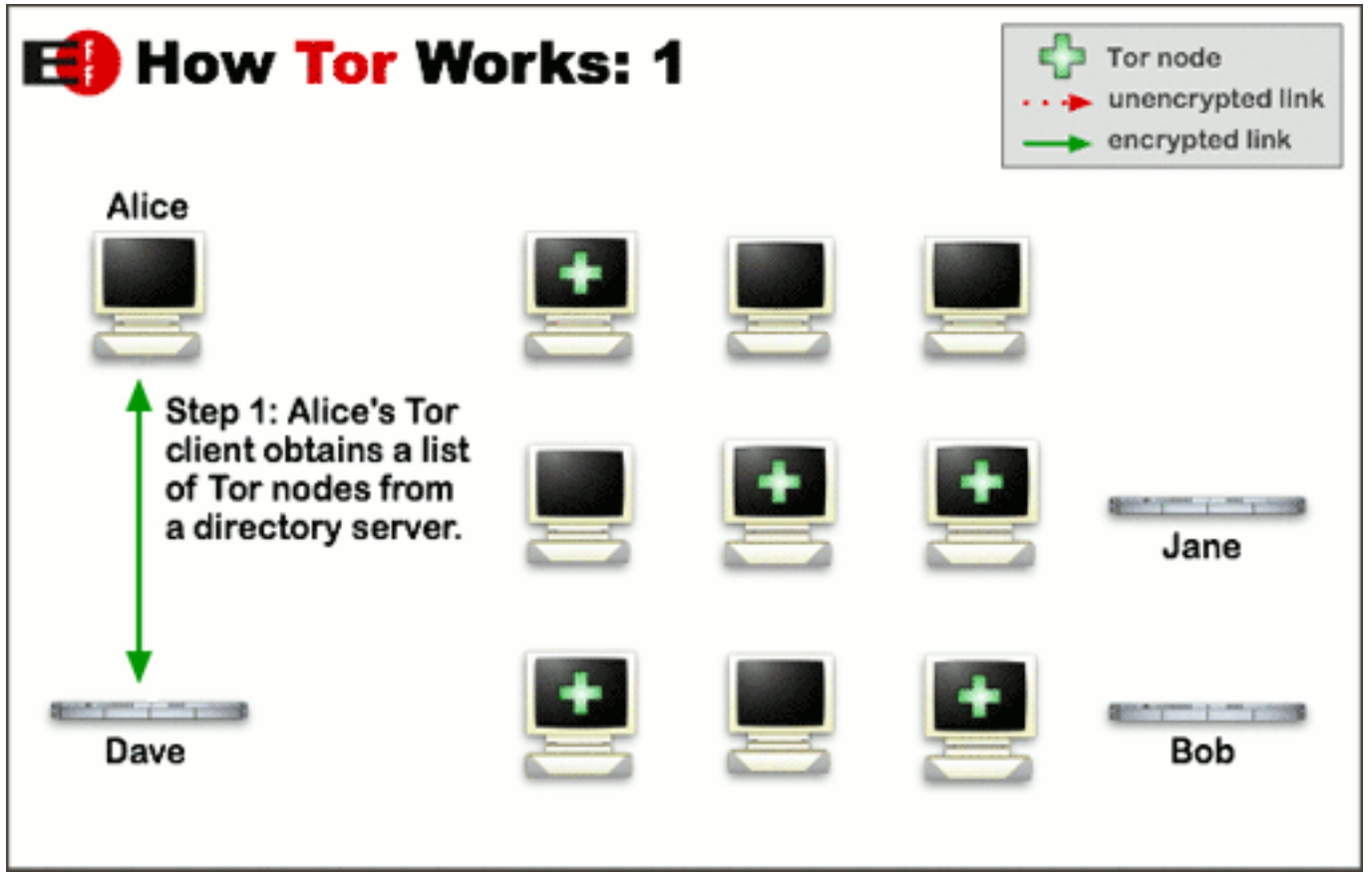
Tor

- The Onion Router
- **Sends data through virtual circuit (3 random relays) – anonymizing communication**
- **Anonymous Internet communication system**
  - SSL / TLS is not enough
  - protect privacy of users
- **Idea ~1995 by US Naval Research Laboratory**
  - Funded by US Naval Research Laboratory and DARPA (Defense Advanced Research Projects Agency)
  - Naval Research Laboratory released the code under free license in 2004
  - EFF began funding development
  - Large project: 495’000 LoC
- **Main use-case: journalists, whistleblowers, and dissidents**
  - Can be used against price discrimination
  - Can be used in the darknet
- **Attention**
  - Tor exit node can see traffic! – Use SSL/TLS
  - Services block Tor exit nodes, e.g., example: Wikipedia
  - Tor exit node operator facing copyright claims - template
  - TCP only (rewrite DNS requests)

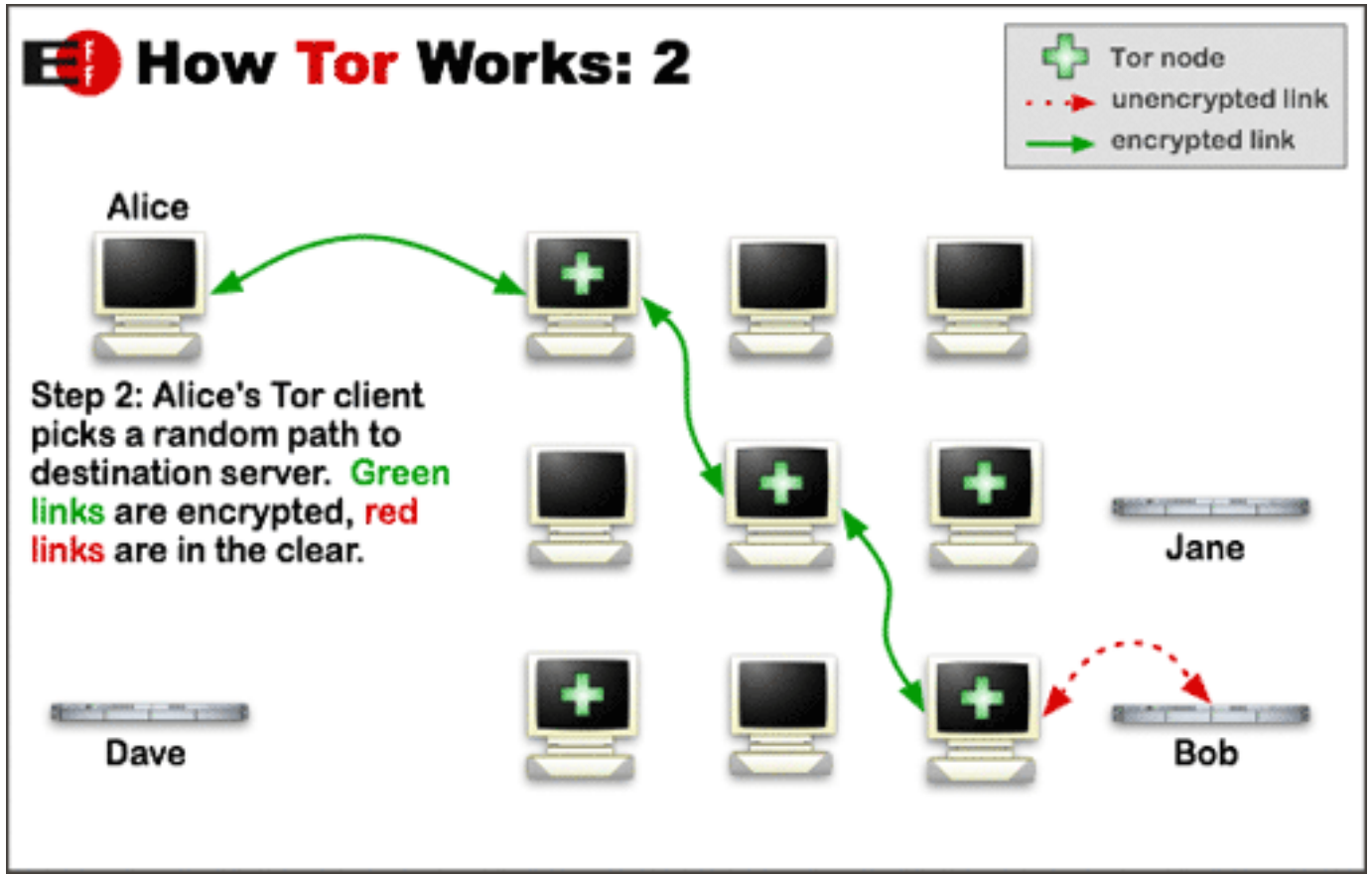


Tor

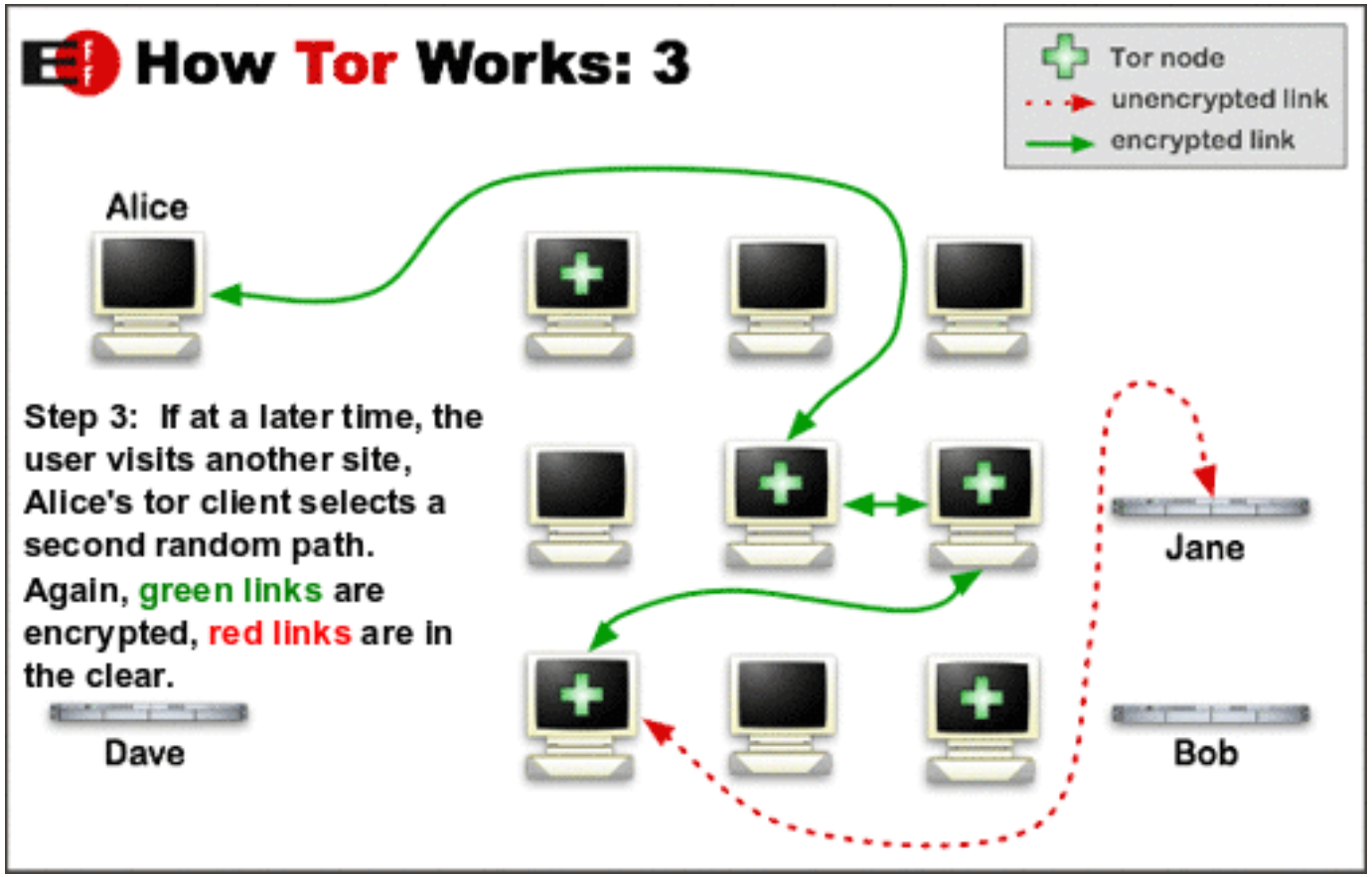
- How it works



- **Alice to Bob**


How Tor Works: 2
Tor node
unencrypted link
encrypted link
Alice
Step 2: Alice's Tor client picks a random path to destination server. Green links are encrypted, red links are in the clear.
Jane
Dave
Bob

- **Alice to Jane**



Tor

- **Onion Services**
  - Hide users locations while offering various services
  - Servers configured to receive connections only through Tor
  - Using DHT

1. **Onion service advertises its existence**
   1. Setup relay – introduction points
2. **Create onion service descriptor, signs with private key, store in DHT: XYZ.onion**
   1. XYZ is 128bit key based on public key
   2. Routing means finding those who are responsible for XYZ
   3. Store descriptor (Introduction points/public key) on those responsible nodes

3. Client gets XYZ.onion

1. Routing means finding those who are responsible for XYZ
2. Get the descriptor on those responsible nodes, connect to introduction points, encrypt with public key

4. Picks rendezvous point, sends initial message encrypted to introduction points

1. Will be forwarded to service endpoint

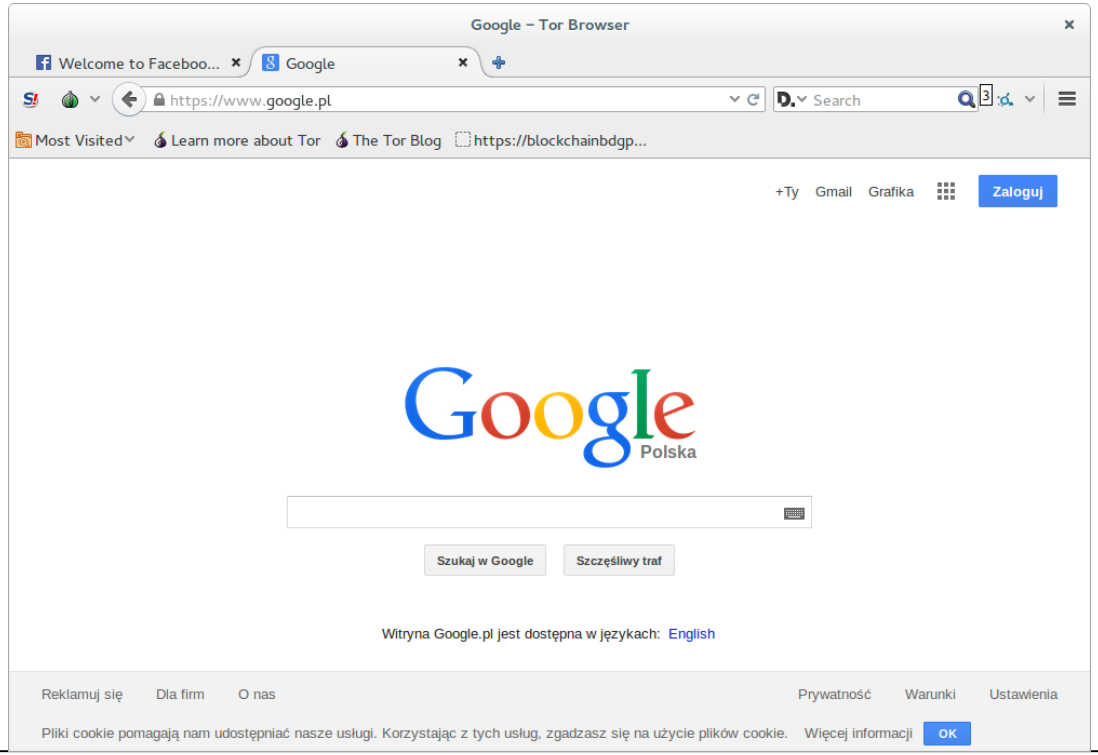
5. Service endpoint opens circuit to rendezvous server

1. User and service endpoint both have 3-relay circuit to rendezvous – 6 relays in total

Tor

- **Data is only encrypted within TOR – if no HTTPS is used, it still can be read**

- **Bomb threats at Harvard**
  - FBI figured out, Tor was used – check who was using Tor in the Harvard network → 2 days later student was caught.



URL: b.socrative.com/student/

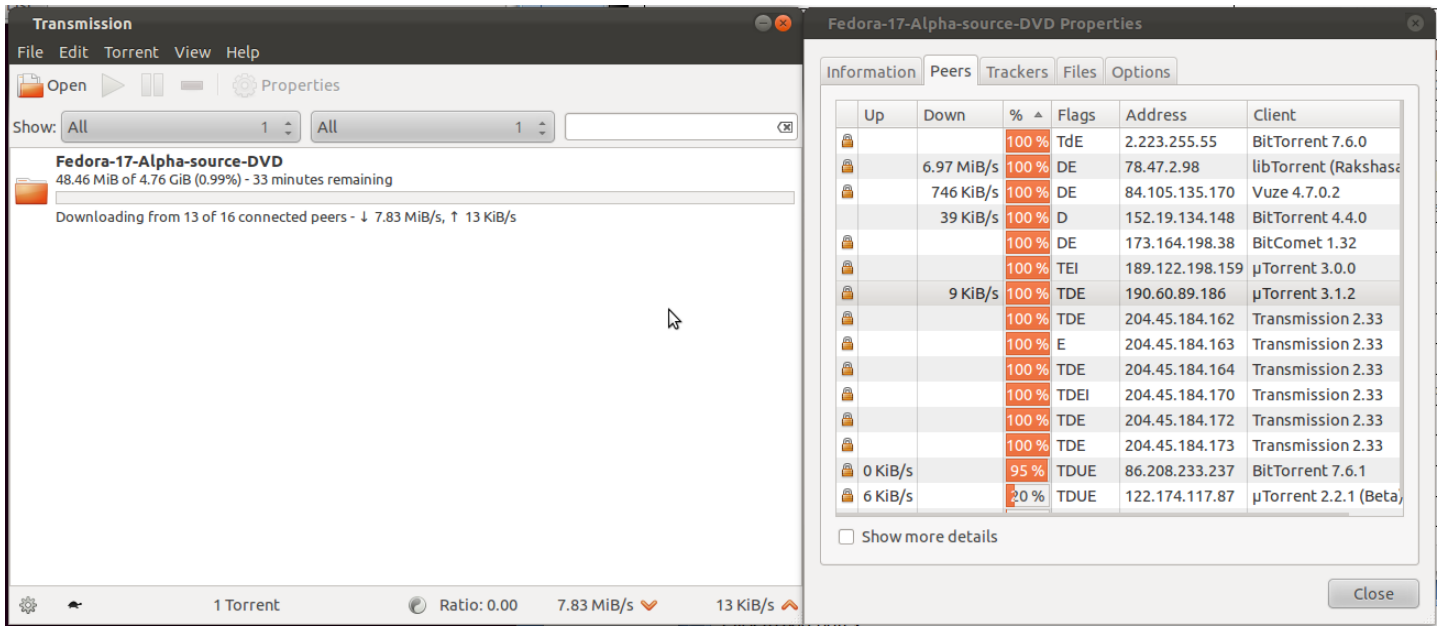
Room: HSR

Introduction

- **BitTorrent**
  - Peer to peer file-sharing system
  - Used to transfer large files
  - Defines a protocol, many clients (Wikipedia):

    ABC, Acquisition, BitComet, BitLet, BitLord, BitTornado, BitTorrent, BitTyrant, Blog Torrent, Deluge, FlashGet, Free Download Manager, Kget, Ktorrent, LimeWire, Meerkat Bittorrent Client, Miro, MLDonkey, µTorrent, OneSwarm, Opera, qBittorent, rTorrent, Shareaza, SymTorrent, Tomato Torrent, Tonido Torrent, Torrent Swapper, TorrentFlux, Transmission, Tribler, Vuze (formerly Azureus), Wyzo, ZipTorrent

- **E.g. Transmission**



Mesh-based File Sharing – BitTorrent

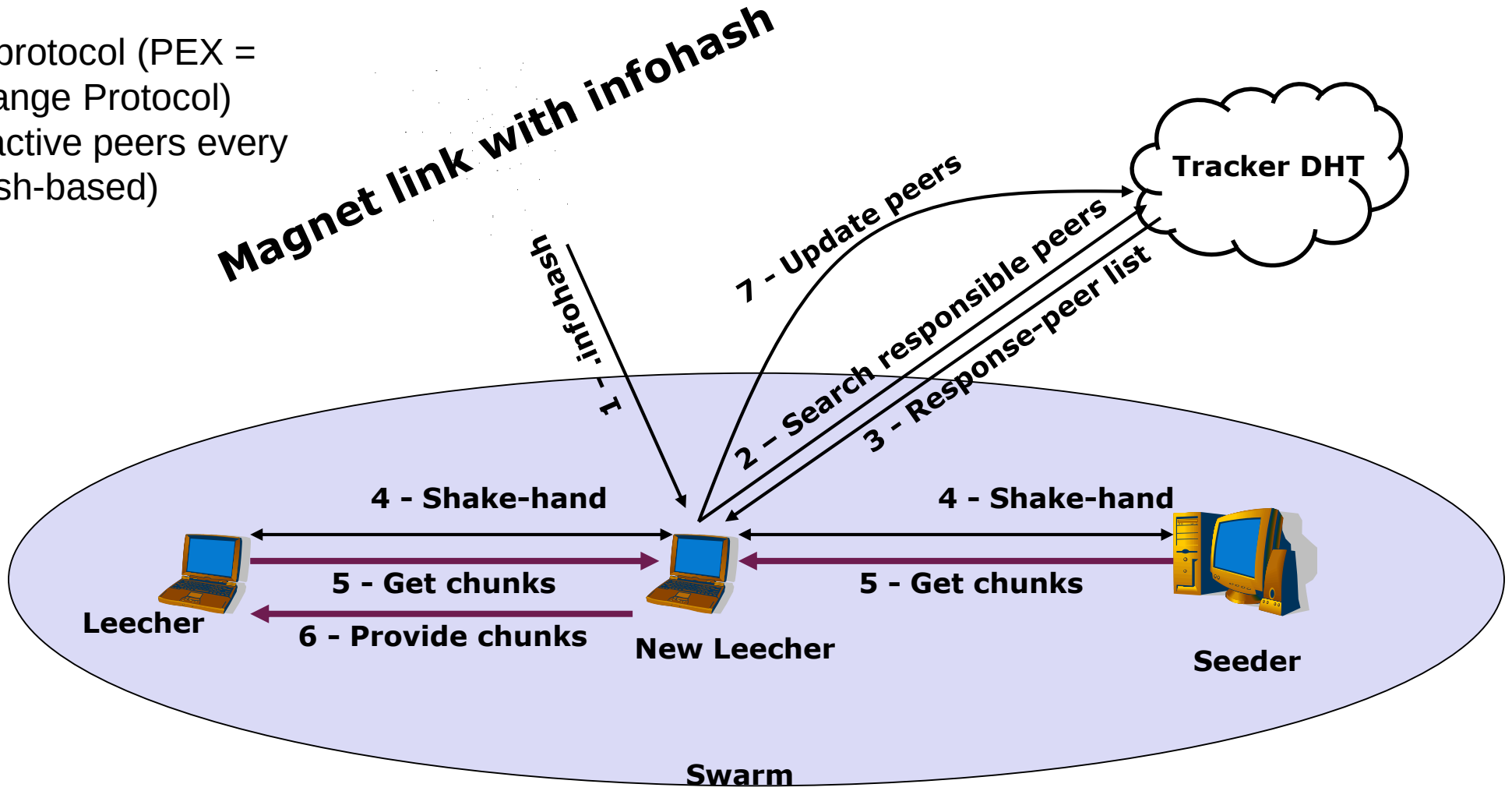
- **Involved peers / roles:**
- **Tracker**
  - Non-content-sharing node
  - Actively tracks
    - Swarm: all peers (seeders and leeches), status of downloaded data - volume
  - Decentralized (DHT), was centralized (HTTP)
- **Seeders**
  - Complete copies and is uploading to other peers
- **Leechers**
  - Incomplete copies of the desired content
  - Download missing pieces
- **Swarm**
  - Sum of all leechers and seeders, participating in a particular torrent
  - One swarm per torrent

- **.torrent info file (ubuntu)**
  - http://releases.ubuntu.com/19.04/ubuntu-19.04-desktop-amd64.iso.torrent
  - magnet:?xt=urn:btih:e84213a794f3ccd890382a54a64ca68b7e925433&dn=Ubuntu+18.04.1+Desktop+64+bit&tr=udp://tracker.le…
- **Contains: name, size, date of files, number of pieces and hash of all pieces, seed+leech**
  - File is split into chunks, e.g, 256kB
  - Bencode (last lecture):

```
i<number>e -> i100e
<length>:<contents> -> 4:test
l<contents>e -> l4:testi100ee
   (two values)
d<contents>e -> d4:testi100ee
   (key test, value 100)
```

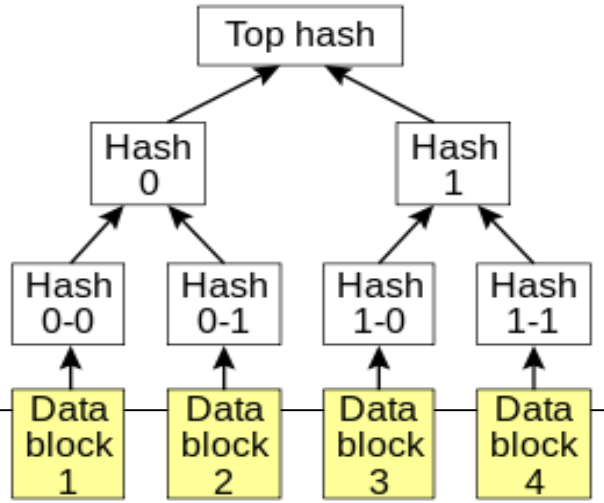
BitTorrent: A Scenario

Gossiping protocol (PEX = Peer Exchange Protocol)
Distribute active peers every minute (push-based)



BitTorrent: Mechanisms

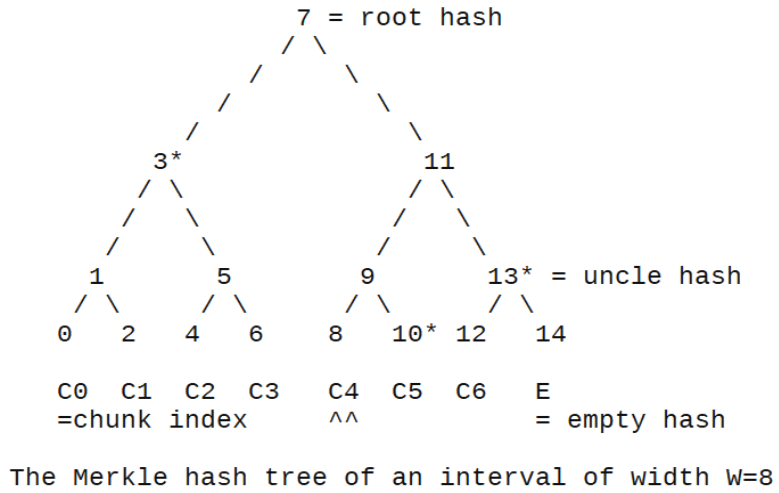
- **Exchange strategy: tit-for-tat**
  - If I give you – you give me
    - Symmetric interest in a swarm
    - Encourages peers to contribute
      (early Gnutella showed that majority does not share anything)
  - Choked: “not sending right now” - connection not being used to transmit
  - Unchoking: best download rates, evaluated every 10sec
  - Optimistic unchoking: rotate every 30sec
  - BitThief!
- **Exchange strategy: rarest-first or random**
  - Good distribution of pieces - in case seeders go offline
- **Magnet links**
  - General purpose, not only for BT
  - `magnet:?xl=1000&dn=song1.mp3&xt=urn:tree:tiger:2A3B…`
- **Hash Tree**
  - Tree of hashes (|| → concatenation)
  - `hash 0 = hash( hash 0-0 || hash 0-1 )`
  - `hash 1 = hash( hash 1-0 || hash 1-1 )`
  - `Top hash = hash( hash 0 || hash 1 )`



http://en.wikipedia.org/wiki/Hash_tree

BitTorrent: Mechanisms

- **Verification of downloaded piece**
  - Peer A has top hash (root hash)
  - Peer downloads C4 from peer B
    - create hash 8
  - Need hash 10, 13, 3 (uncle hash)
    - Can be from peer B
  - With 8,10,13,3 can create/check root hash



The Merkle hash tree of an interval of width W=8

- **µTP**
  - Micro Transport Protocol - application-layer congestion-control protocol using UDP at the transport-layer
  - Congestion (queue) detected by time rather than packet loss (TCP)
  - TCP friendly (LEDBAT), background traffic

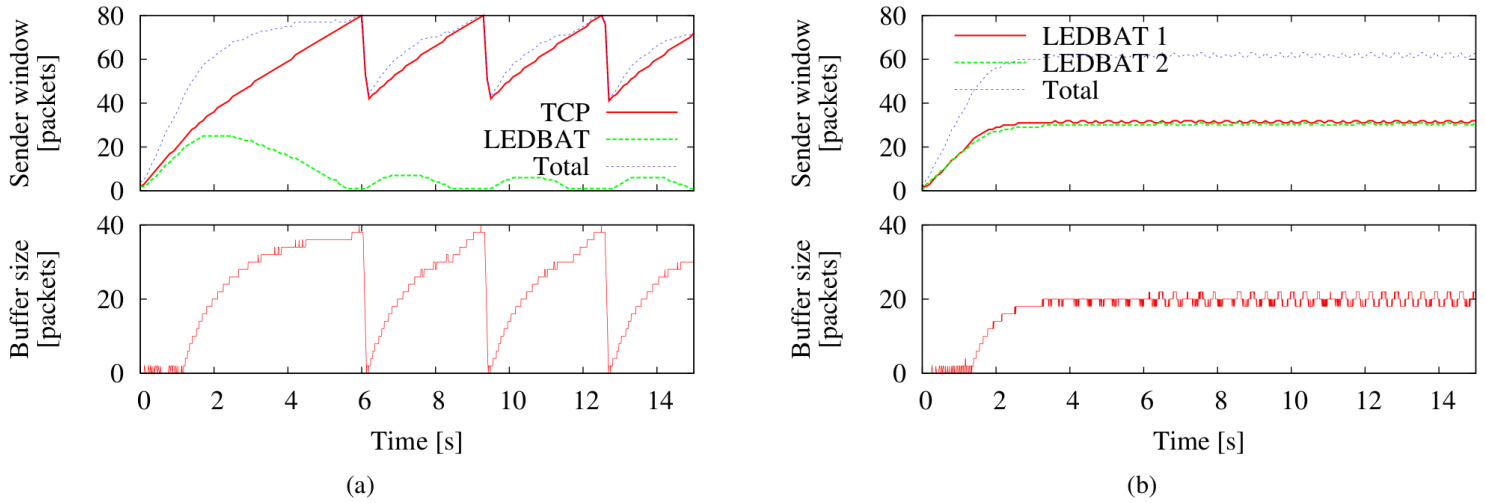


Fig. 2. Temporal evolution of the sender window (top) and of the queue size (bottom) for TCP-LEDBAT (a) and LEDBAT-LEDBAT interaction (b)

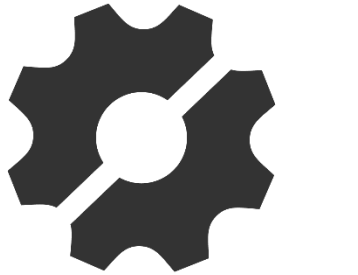
https://datatracker.ietf.org/doc/draft-ietf-ppsp-peer-protocol/
Section 5.2

URL: b.socrative.com/student/

Room: HSR

Message Queue Use Case: ICOnator Project

- **“An easy, secure, configurable, and scalable open source ICO engine – supporting the world's tokenization”**
- **https://iconator.io, https://github.com/ICOnator**
- **4 core components: core, rates, email, monitor**
  - Initial release, one big application
  - Issue under high load
  - Idea: decouple components with message queues
- **Easy deployment to Cloud providers**
  - Amazon AWS, Heroku, Swisscom AppCloud, Pivotal
  - Increases complexity! But more scalable
- **ICO: fundraising mechanism in which new projects sell their own crypto tokens in exchange for bitcoin and ether**

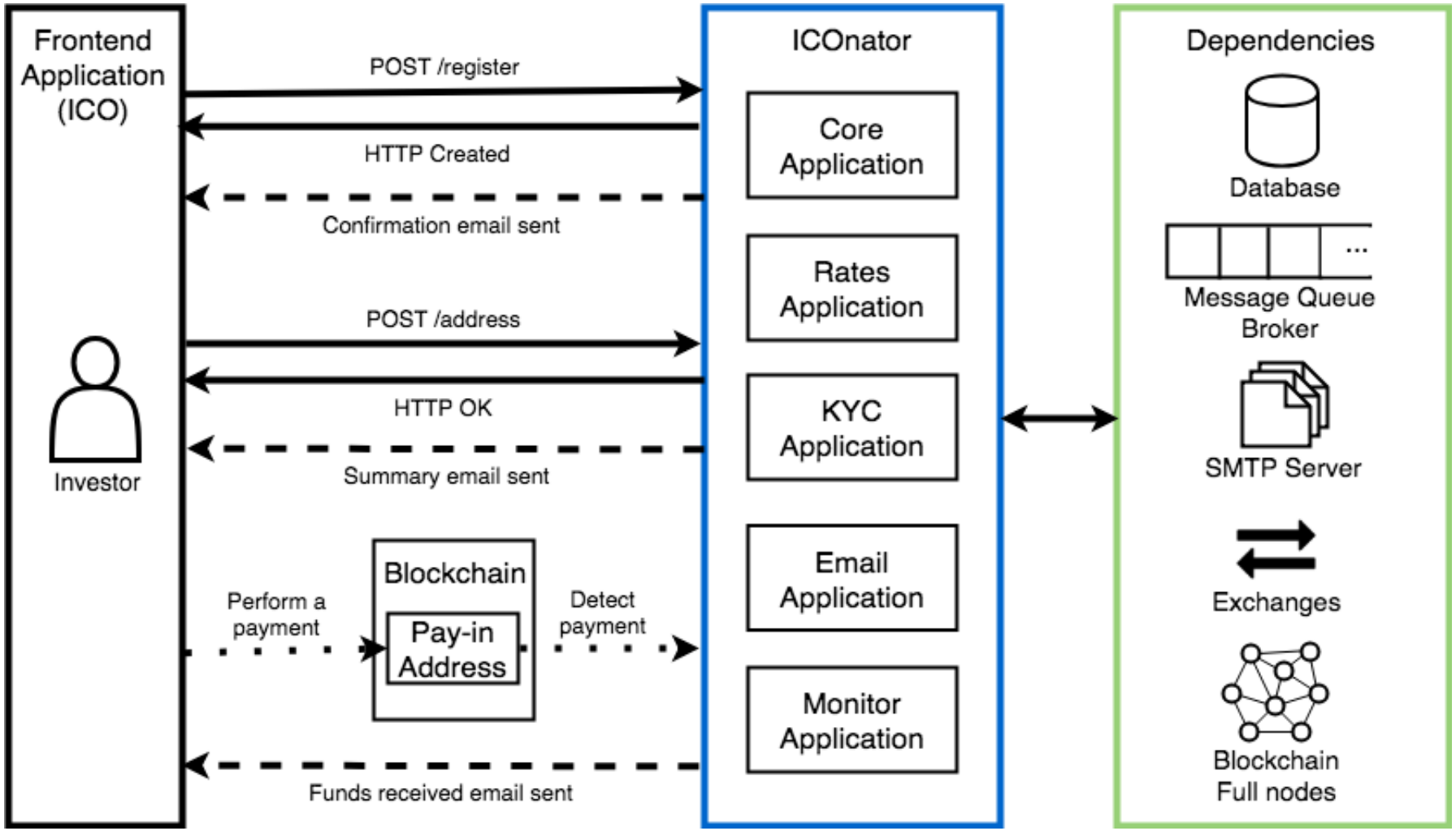


ICONATOR

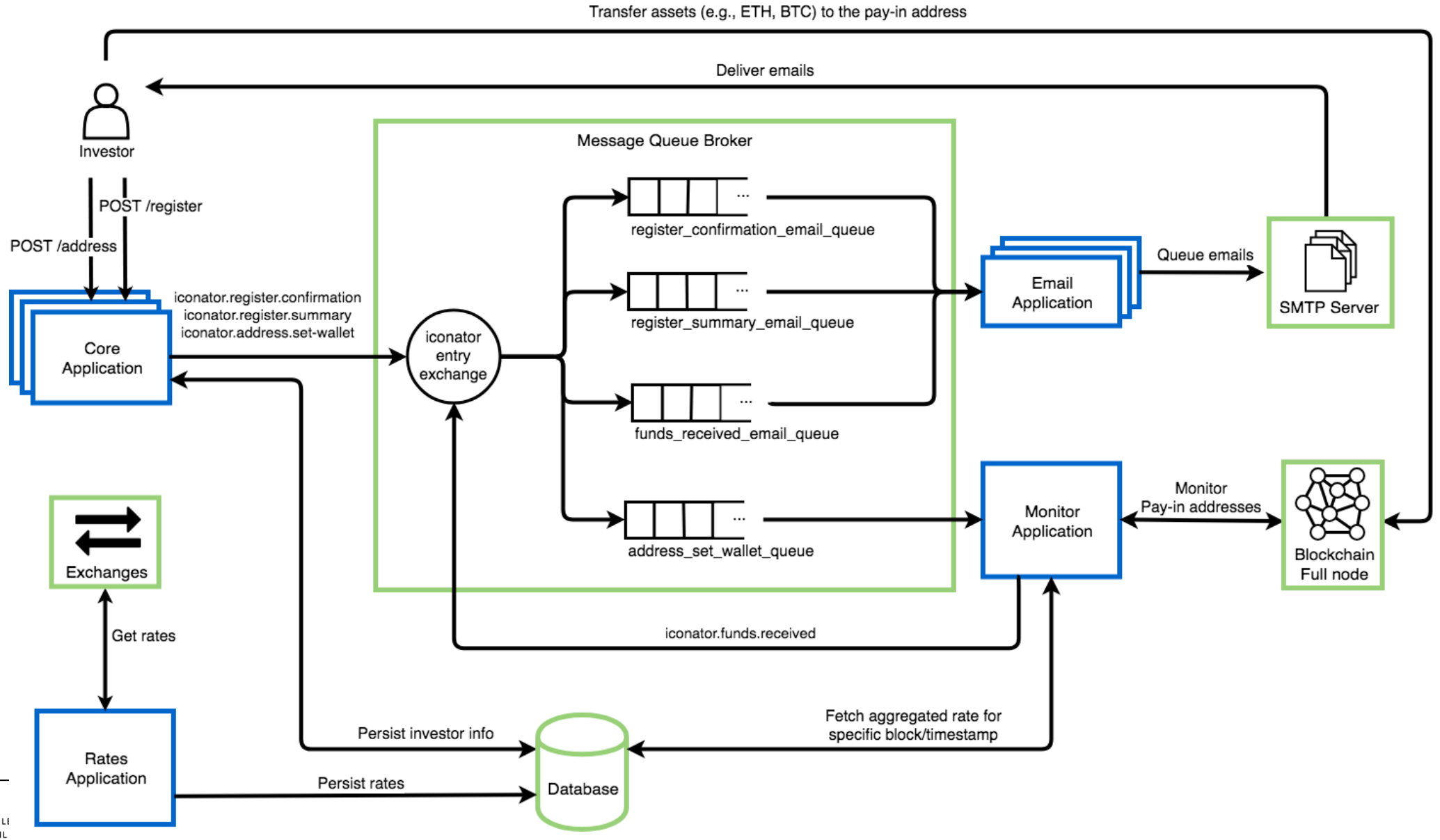
powered by

ICO NATOR

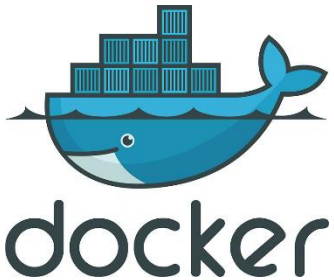
High Level Architecture



Lower Level Architecture



Software Development Technologies



URL: https://qfeedback.hsr.ch/q-feedback/login

Q-Feedback

Questions?

Prof. Dr. Thomas Bocek
thomas.bocek@hsr.ch