HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

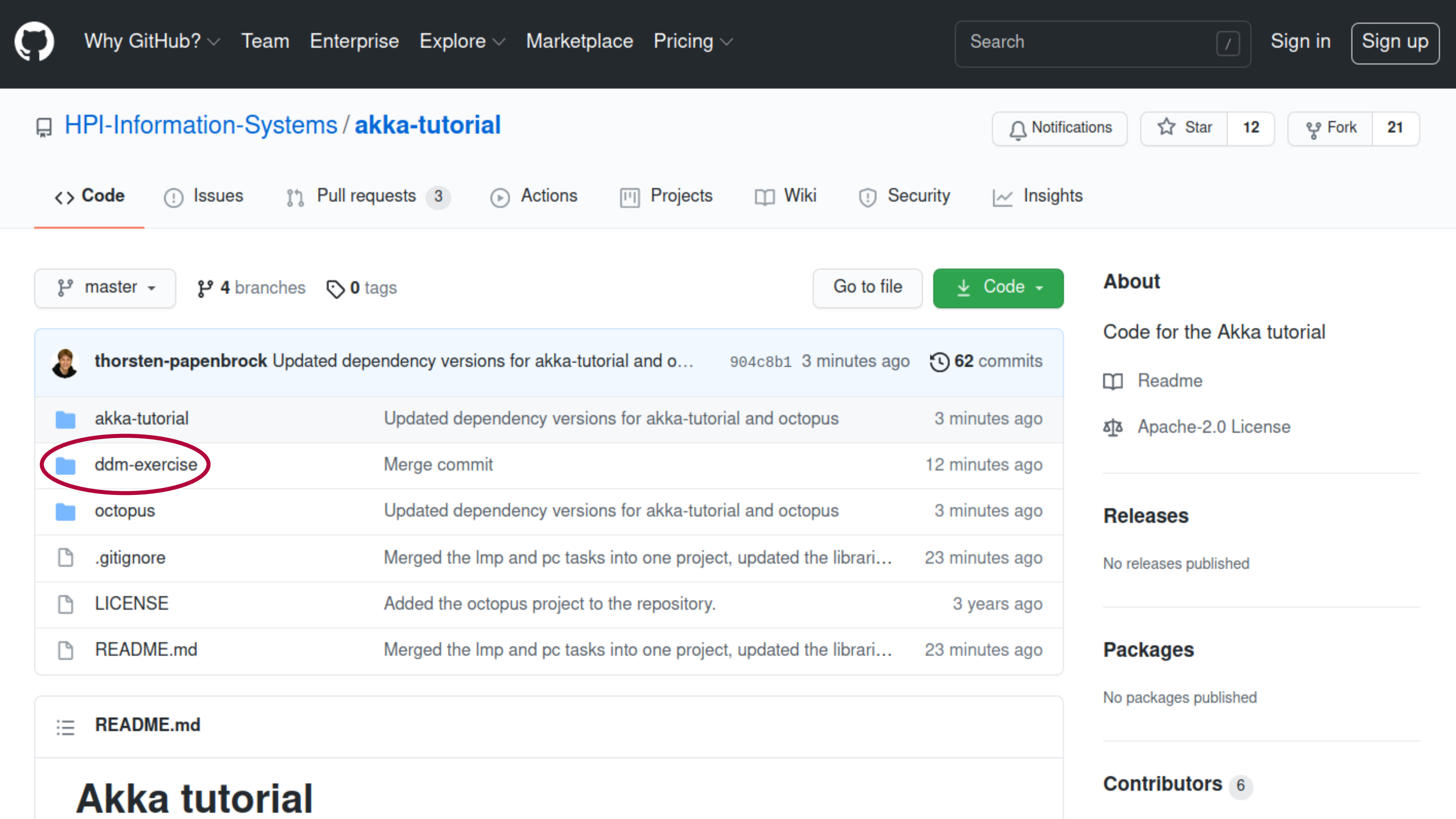
# Distributed Data Management
# Exercise 1 Evaluation

Thorsten Papenbrock

F-2.04, Campus II

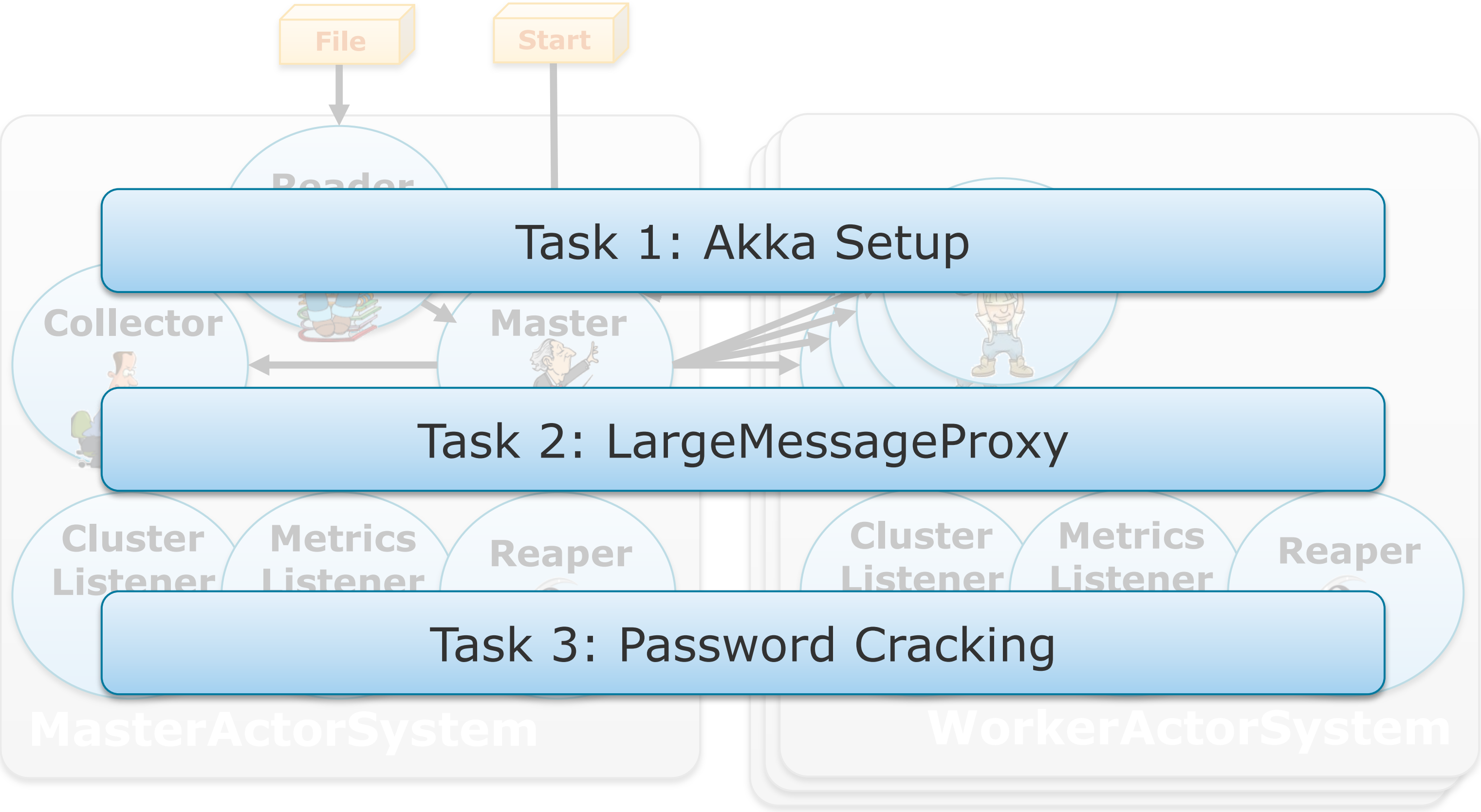
Hasso Plattner Institut


akka
“I wait for green”
“Road ahead is free!”
“I wait for crossing traffic”
“I accelerate!”
“You are not in my path!”
“Attention, I break!”


Why GitHub?
Team
Enterprise
Explore
Marketplace
Pricing
Search
Sign in
Sign up
HPI-Information-Systems / akka-tutorial
Notifications
Star 12
Fork 21
Code
Issues
Pull requests 3
Actions
Projects
Wiki
Security
Insights
master
4 branches
0 tags
Go to file
Code
thorsten-papenbrock Updated dependency versions for akka-tutorial and o…
904c8b1 3 minutes ago
62 commits
akka-tutorial
Updated dependency versions for akka-tutorial and octopus
3 minutes ago
ddm-exercise
Merge commit
12 minutes ago
octopus
Updated dependency versions for akka-tutorial and octopus
3 minutes ago
.gitignore
Merged the lmp and pc tasks into one project, updated the librari…
23 minutes ago
LICENSE
Added the octopus project to the repository.
3 years ago
README.md
Merged the lmp and pc tasks into one project, updated the librari…
23 minutes ago
README.md
Akka tutorial
About
Code for the Akka tutorial
Readme
Apache-2.0 License
Releases
No releases published
Packages
No packages published
Contributors 6

Homework

# ddm-exercise




Task 1: Akka Setup

Task 2: LargeMessageProxy

Task 3: Password Cracking

Homework

# ddm-exercise


**Distributed Data Management**

Akka Actor Programming

Homework

# Task 1 – Akka Setup

1. Form teams of two students.
2. Create a public GitHub repository.
3. Copy or fork the ddm-exercise project from the exercise repository https://github.com/HPI-Information-Systems/akka-tutorial into your repository.
4. Build, understand and test the ddm-exercise project.
5. Optional: Check out and play with the akka-tutorial and octopus projects.
6. Send your first and last names, a group name and the link of your repository via email to: thorsten.papenbrock@hpi.de

## Homework

# Task 1 – Teams

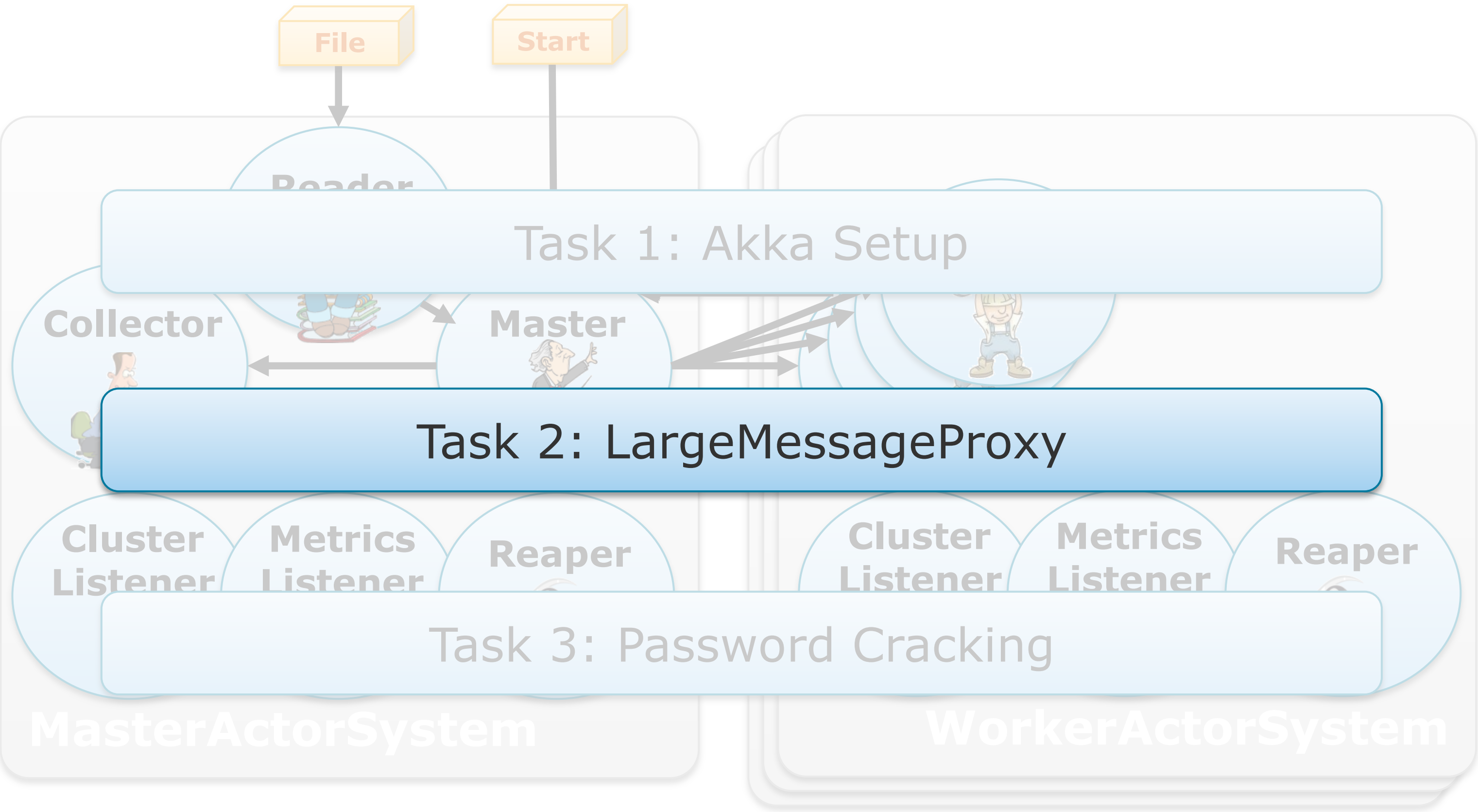
| Team | Task 2 passed? | Task 3 passed? |
|---|---|---|
| supreme-broccoli | | |
| w00t? | | |
| Code Monkeys | | |
| ddm_team_42 | | |
| Duftes Daten Mischen (DDM) | | |
| Dally | | |
| Distributed Wealth | | |
| the_reapers | | |
| Chewbakka | | |
| BlockchainOnAkka | | |
| Unknown Pleasures | | |
| Taube_Nuesschen | | |
| So Called Engineers | | |
| Alpha | | |
| Euphorische Elefanten | | |
| mAKKAronis | | |
| MeMyselfAndI | | |
| Multiprocessing Moguls | | |
| AlpAkka | | |
| DeadlyThread | | |

Homework

# Task 1 – Teams

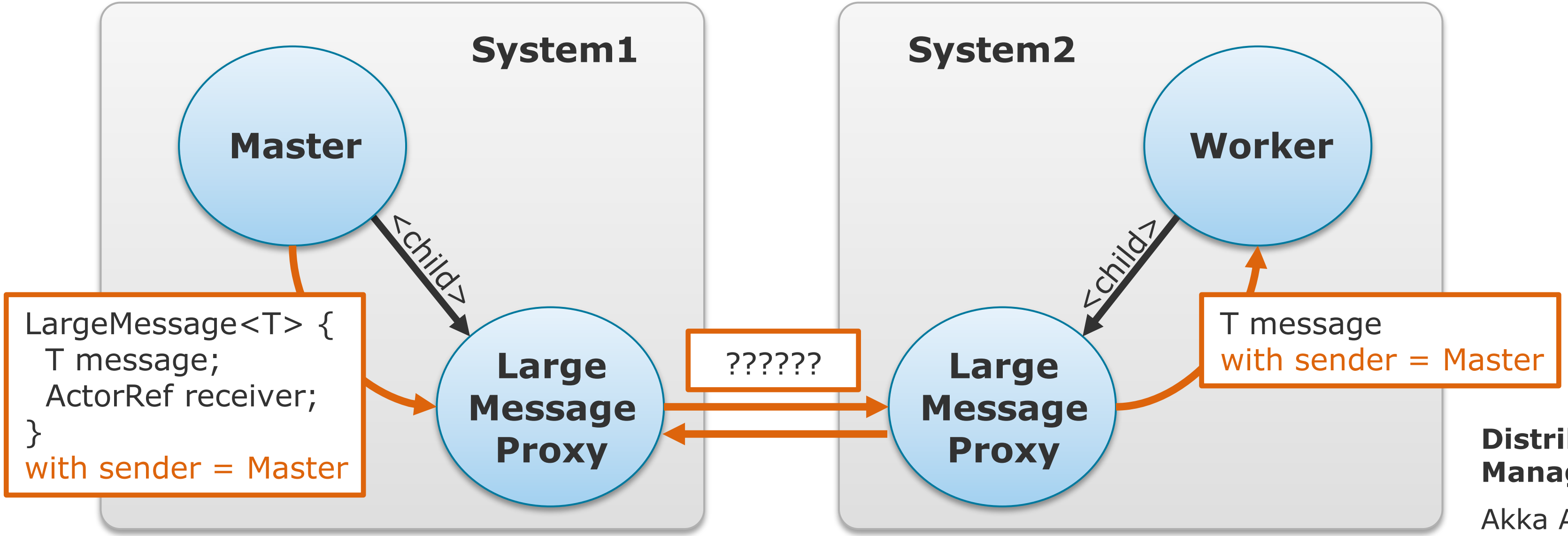
| Team | Task 2 passed? | Task 3 passed? |
|---|---|---|
| supreme-broccoli | Yes | Yes |
| w00t? | Yes | Yes |
| Code Monkeys | Yes | Yes |
| ddm_team_42 | Yes | Yes |
| Duftes Daten Mischen (DDM) | Yes | Yes |
| Dally | Yes | Yes |
| Distributed Wealth | Yes | Yes |
| the_reapers | Yes | Yes |
| Chewbakka | Yes | Yes |
| BlockchainOnAkka | Yes | Yes |
| Unknown Pleasures | Yes | Yes |
| Taube_Nuesschen | Yes | Yes |
| So Called Engineers | Yes | Yes |
| Alpha | Yes | Yes |
| Euphorische Elefanten | Yes | Yes |
| mAKKAronis | Yes | Yes |
| MeMyselfAndI | Yes | Yes |
| Multiprocessing Moguls | Yes | Yes |
| AlpAkka | Yes | Yes |
| DeadlyThread | Yes | Yes |

# Homework
## ddm-exercise



Homework

# Task 2 – LargeMessageProxy




Task

- Implement the LargeMessageProxy actor!

```java
@Override
public Receive createReceive() {
    return receiveBuilder()
            .match(LargeMessage.class, this::handle)
            .match(BytesMessage.class, this::handle)
            .matchAny(object -> this.log().info("Received unknown message: \"{}\"", object.toString()))
            .build();
}

private void handle(LargeMessage<?> largeMessage) {
    Object message = largeMessage.getMessage();
    ActorRef sender = this.sender();
    ActorRef receiver = largeMessage.getReceiver();
    ActorSelection receiverProxy = this.context().actorSelection(receiver.path().child(DEFAULT_NAME));

    // TODO: Implement a protocol that transmits the potentially very large message object.
    // The following code sends the entire message wrapped in a BytesMessage, which will definitely fail in a distributed setting if the message is large!
    // Solution options:
    // a) Split the message into smaller batches of fixed size and send the batches via ...
    //    a.a) self-build send-and-ack protocol (see Master/Worker pull propagation), or
    //    a.b) Akka streaming using the streams build-in backpressure mechanisms.
    // b) Send the entire message via Akka's http client-server component.
    // c) Other ideas ...
    // Hints for splitting:
    // - To split an object, serialize it into a byte array and then send the byte array range-by-range (tip: try "KryoPoolSingleton.get()").
    // - If you serialize a message manually and send it, it will, of course, be serialized again by Akka's message passing subsystem.
    // - But: Good, language-dependent serializers (such as kryo) are aware of byte arrays so that their serialization is very effective w.r.t.
    //   serialization time and size of serialized data.
    receiverProxy.tell(new BytesMessage<>(message, sender, receiver), this.self());
}

private void handle(BytesMessage<?> message) {
    // TODO: With option a): Store the message, ask for the next chunk and, if all chunks are present, reassemble the message's content, deserialize it and
    //       pass it to the receiver.
    // The following code assumes that the transmitted bytes are the original message, which they shouldn't be in your proper implementation ;-)
    message.getReceiver().tell(message.getBytes(), message.getSender());
}
```

# Approach

**Master/Worker pull-protocol?**

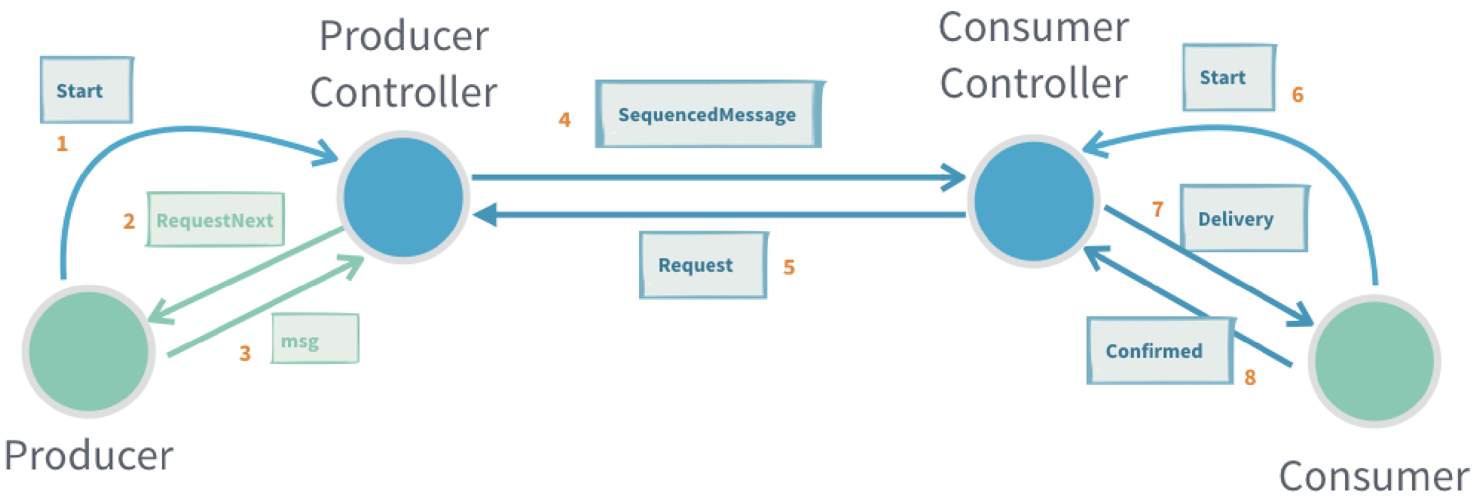
**Akka Streams?**

**Akka Client-Server?**

**Alternative approach?**

# Approach – Point-to-Point Pattern

The Point-to-Point pattern has support for **automatically splitting up large messages and assemble them again on the consumer side**. This feature is useful for avoiding head of line blocking from serialization and transfer of large messages.

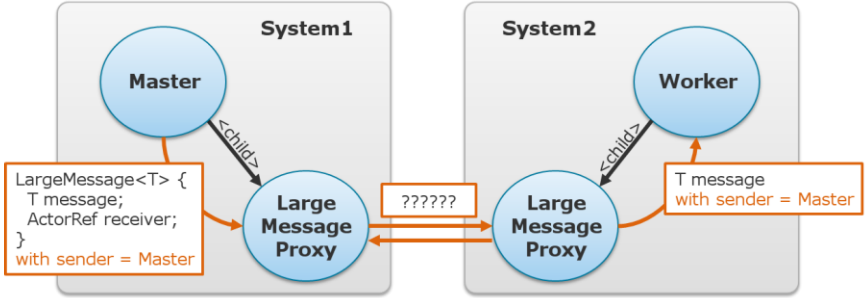



https://doc.akka.io/docs/akka/current/typed/reliable-delivery.html

Homework

# Task 2 – LargeMessageProxy

## Rules




- Do not mess with the time measurement: It should start with the registration time and it should end when receiving the data.
- Do not change the command line interface or app name; otherwise, the automatic test scripts will fail.
- Do not change the LargeMessage class; the LargeMessageProxy should be able to send messages of any type T.
- Use maven to import additional libraries if you need some.
- Do not use the disk.
- Feel free to change everything inside the LargeMessageProxy!

Homework

# Evaluation – Odin/Thor Cluster


Homework

# Task 2 – Test

| Team | Executes? | Works? |
|---|---|---|
| supreme-broccoli | | |
| w00t? | | |
| Code Monkeys | | |
| ddm_team_42 | | |
| Duftes Daten Mischen (DDM) | | |
| Dally | | |
| Distributed Wealth | | |
| the_reapers | | |
| Chewbakka | | |
| BlockchainOnAkka | | |
| Unknown Pleasures | | |
| Taube_Nuesschen | | |
| So Called Engineers | | |
| Alpha | | |
| Euphorische Elefanten | | |
| mAKKAronis | | |
| MeMyselfAndI | | |
| Multiprocessing Moguls | | |
| AlpAkka | | |
| DeadlyThread | | |

Homework

# Task 2 – Test

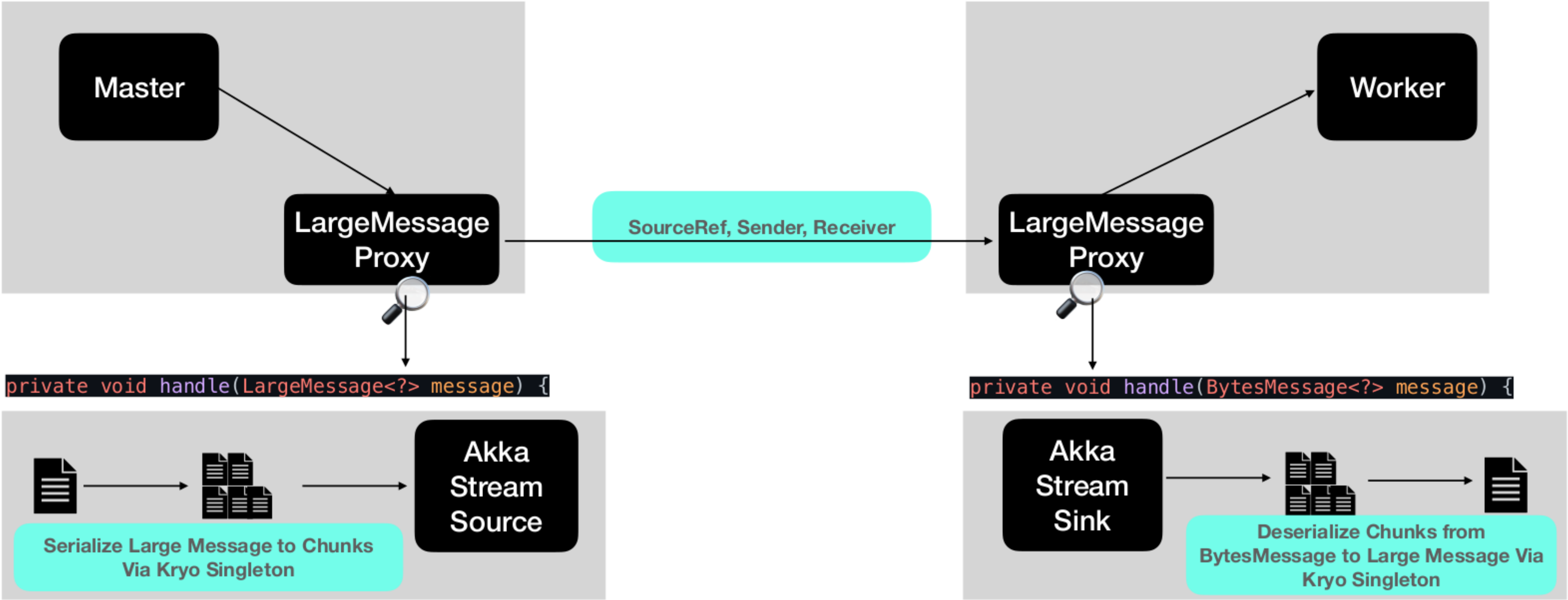
| Team | Executes? | Works? |
| --- | --- | --- |
| supreme-broccoli | Yes | No |
| w00t? | Yes | Yes |
| Code Monkeys | Yes | Yes |
| ddm_team_42 | Yes | Yes |
| Duftes Daten Mischen (DDM) | Yes | Yes |
| Dally | Yes | Yes |
| Distributed Wealth | Yes | Yes |
| the_reapers | Yes | Yes |
| Chewbakka | Yes | Yes |
| BlockchainOnAkka | Yes | Yes |
| Unknown Pleasures | Yes | Yes |
| Taube_Nuesschen | Yes | Yes |
| So Called Engineers | Yes | Yes |
| Alpha | Yes | Yes |
| Euphorische Elefanten | Yes | Yes |
| mAKKAronis | Yes | Yes |
| MeMyselfAndI | Yes | Yes |
| Multiprocessing Moguls | Yes | Yes |
| AlpAkka | Yes | Yes |
| DeadlyThread | Yes | Yes |

```
com.esotericsoftware.kryo.KryoException:
Class cannot be created
(missing no-arg constructor):
akka.stream.impl.streamref.SourceRefImpl
```

**Solution:**
SourceRefImpl cannot be serialized with Kryo. Change the serialization for SourceRefImpl to Java Serializable in the config file and specify kryo for other messages that can actually be serialized with it.

# Assignment 2

# Group: supreme-broccoli 👑🥦




First, we use the Kryo Singleton to serialize the message and chunk it subsequently as a List of ByteStrings.
Then, we create a Source that is a ByteStream from the serialized message chunks and run it with a sourceRef with help from Akka.
Further, this sourceRef is what we send as BytesMessage from one to the other system.
The other system can generate a Sink from this sourceRef.
This Sink then, again with the help from Akka, can be used to stream the entire message from one system to the other.

# Task 1 – Test

## LargeMessageProxy

- LargeMessage serialized with KryoPoolSingleton.get()
- Connect Master with Worker
- Streaming with Sink
  - Creating Batches to stream via Sink
  - Using Sink.actorRefWithBackpressure()
  - StreamSyncMessage -> StreamInitializedMessage -> StreamCompletedMessage (Deserialization)

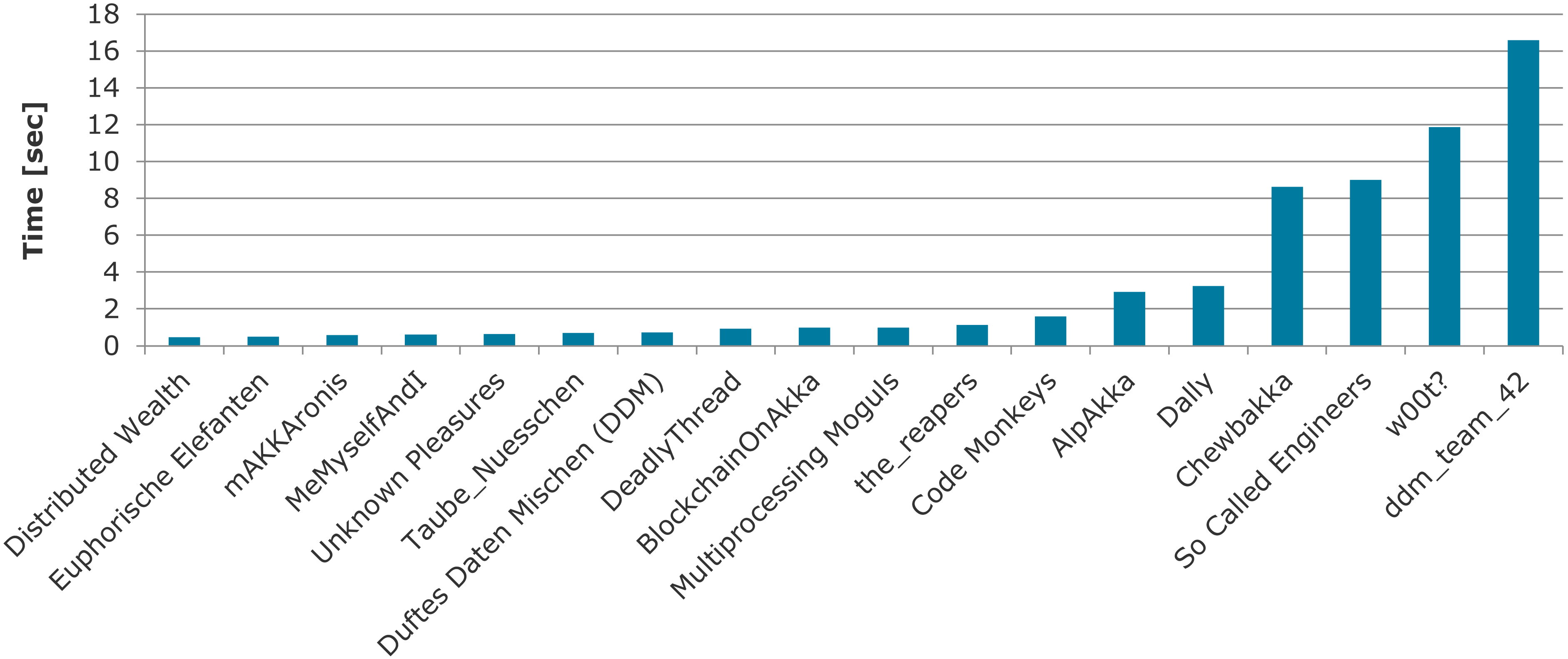
| Team | | |
|---|---|---|
| supreme-bro | | |
| w00t? | | |
| Code Monkey | | |
| ddm_team_4 | | |
| Duftes Daten | | |
| Dally | | |
| Distributed W | | |
| the_reapers | | |
| Chewbakka | | |
| BlockchainOr | | |
| Unknown Ple | | |
| Taube_Nuess | | |
| So Called Eng | | |
| Alpha | Yes | Yes |
| Euphorische Elefanten | Yes | Yes |
| mAKKAronis | Yes | Yes |
| MeMyselfAndI | Yes | Yes |
| Multiprocessing Moguls | Yes | Yes |
| AlpAkka | Yes | Yes |
| DeadlyThread | Yes | Yes |

Submitted jar file did not work, but I later figured out that the code worked; unfortunately the cluster time was up, so I could not produce further experimental results.

Assignment 2

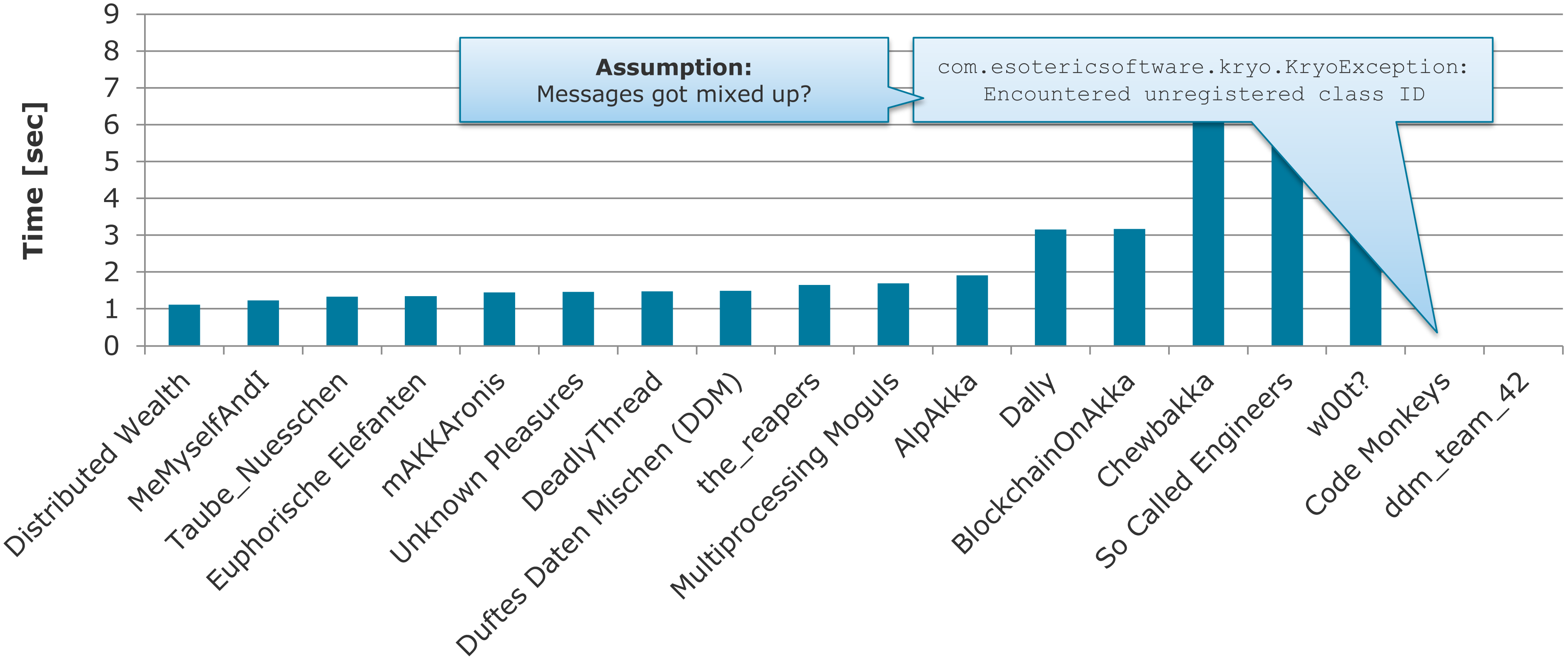
# 1 master, 1 worker à 1 worker, WMS 10MB

Hasso Plattner Institut



Assignment 2

# 1 master, 1 worker à **10** worker, WMS 10MB



# Algorithmus - Large Message Proxy

Der LargeMessageProxy hat drei Aufgaben:

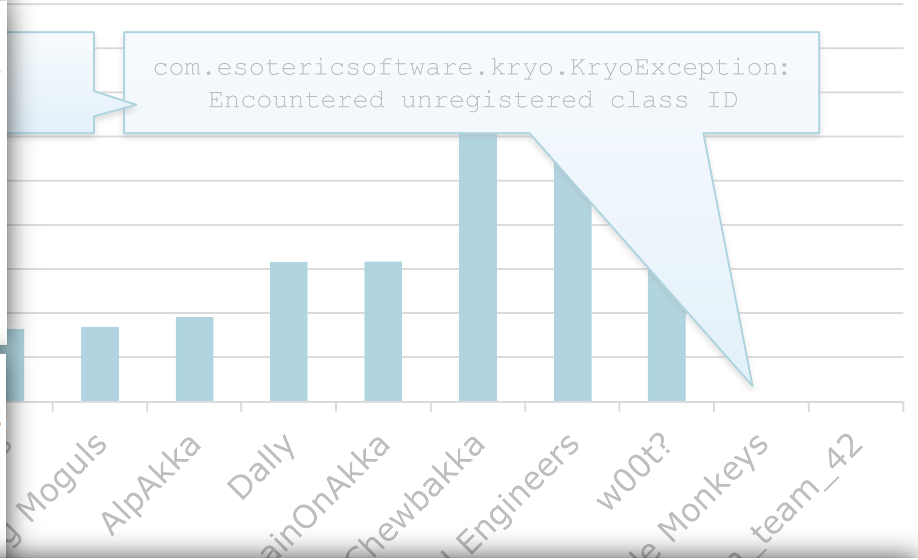
1. Aufteilen von großen LargeMessages in kleine ByteMessages
2. Zusammenfügen kleiner ByteMessages zu einer LargeMessage
3. Kommunikation mit zweiter LargeMessageProxy

# Funktionsweise Sender

1. Serialisieren der LargeMessage in bytes
2. Aufteilen der bytes in gleichgroße Blöcke (a 512 byte, konfigurierbar)
3. Zwischenspeichern der Blöcke als BytesMessage
4. Senden an den ReceiverProxy
    a. Sende eine BytesMessage. Ein Flag gibt dem Receiver an ob es sich um die letzte Bytesmessage handelt
    b. Empfange Empfangsbestätigung
    c. Wiederhole bei 4.a)

# Funktionsweise Empfangen

1. Empfangen aller BytesMessages
    a. Empfange BytesMessage
    b. Speichere in Liste
    c. Wenn das Flag angibt, dass noch weitere Messages übrig sind, sende Empfangsbestätigung, gehe zu 1.a)
2. Extrahiere die Datenblöcke aus den BytesMessages
3. Füge sie zu einem ByteArray zusammen
4. Deserialisiere die Bytes zur ursprünglichen LargeMessage

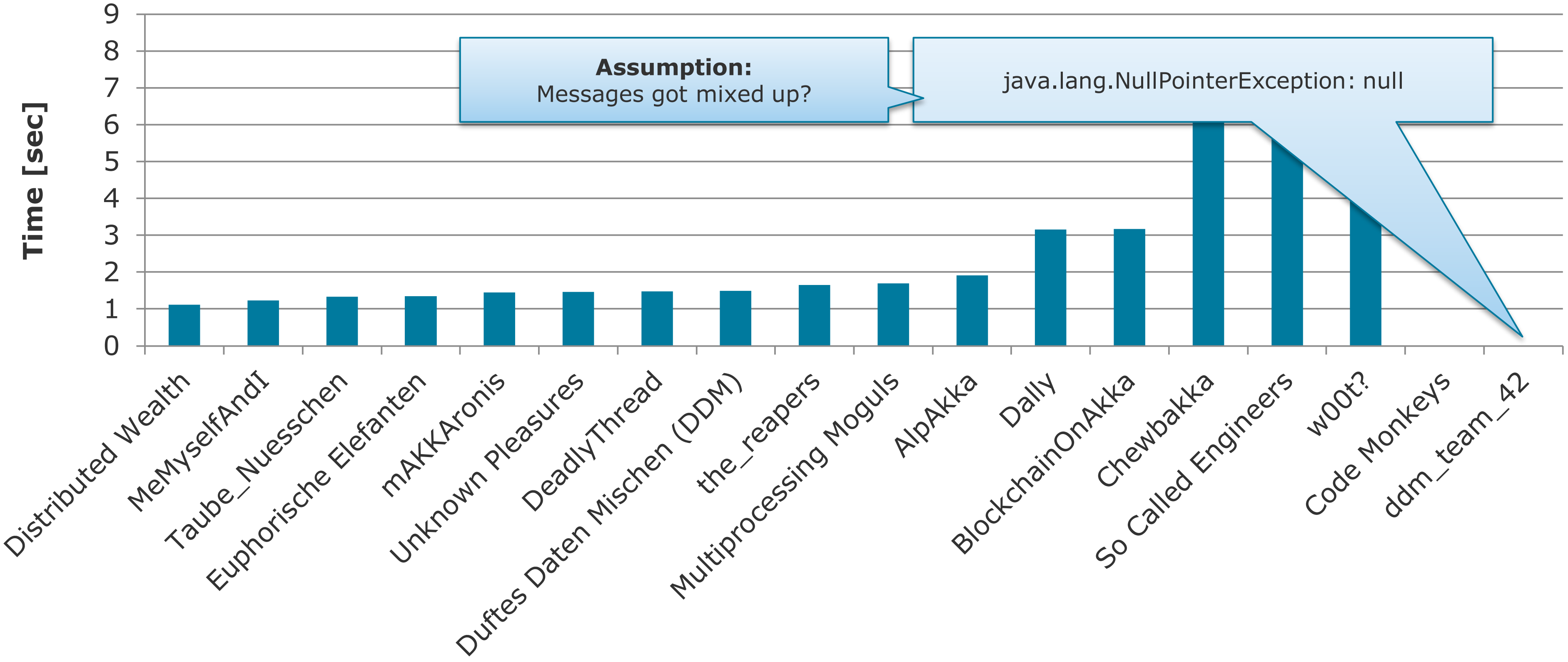



# Funktionsweise

Durch das Bestätigen des Empfängers der einzelnen BytesMessages werden diese nacheinander, in der richtigen Reihenfolge in der Empfangsliste gespeichert. Das ist notwendig, um die ursprüngliche LargeMessage wiederherstellen zu können.

Assignment 2

# 1 master, 1 worker à **10** worker, WMS 10MB



**Assignment 2 (LargeMessageProxy)**

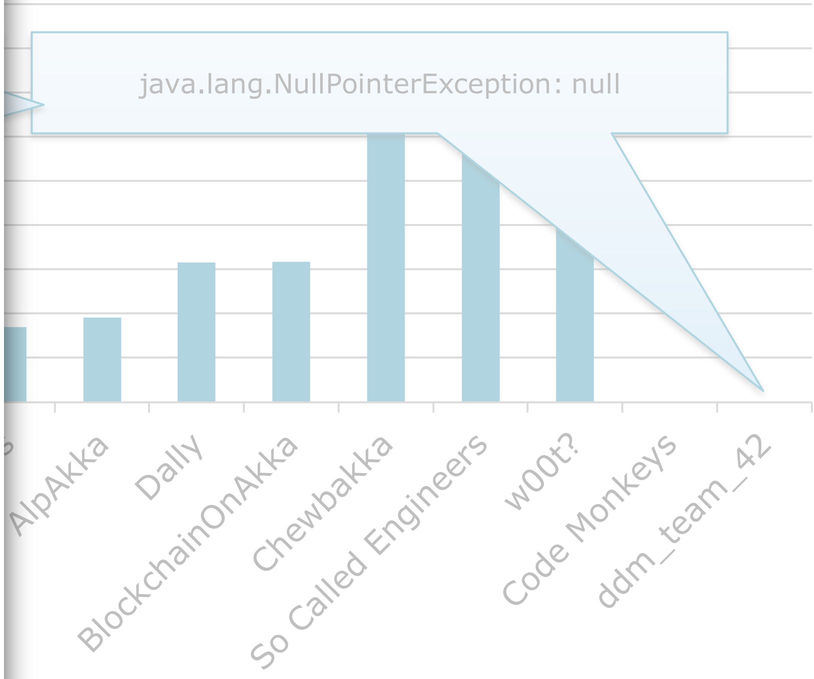
1. In *handle(LargeMessage<?> largeMessage),* the sending LargeMessageProxy receives a large message and converts it into a byte array using Kryo. After this, the message hash is calculated as well as its length. With the message length and the preset parameter *MESSAGE_SIZE* the number of message chunks to transmit is calculated.
   Thereafter, the message is split into chunks and each chunk is transmitted to the receiving LargeMessageProxy via Akka's tell()-Method.

2. The Serializable *BytesMessage<T>* was extended by the hash and the length of the complete message, the number of the transmitted BytesMessage and the number of total messages.

3. In *handle(BytesMessage<?> message)* the receiving LargeMessageProxy reassembles the received message chunks. Received LargeMessages are stored in the HashMap *messagesToReceiveMap*. The message hash is used as key. If this map doesn't exist, the LargeMessageProxy instantiates it. For each transmitted message, the messageToReceiveMap contains a TreeMap storing the message chunks.
   Thereupon the LargeMessageProxy checks if the message associated with the received chunk already has an entry in the messagesToReceiveMap and creates it otherwise. A TreeMap is used so the chunks will be sorted automatically.
   A reference to this TreeMap is stored as *chunkMap*.

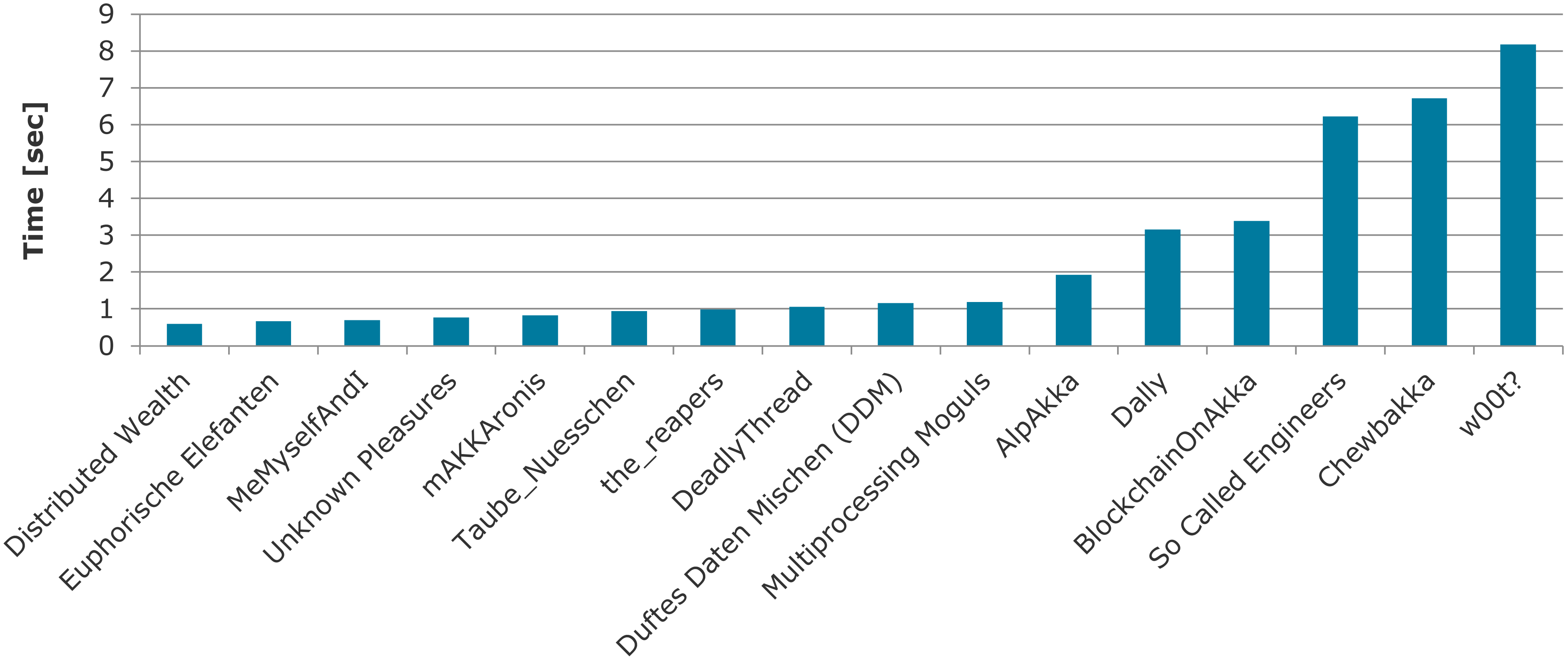
4. Now the receiver proxy stores the received chunk in the *chunkMap*. The used key is the message number. First it checks if the chunk does already exist. If the size of the chunkMap equals the number of chunks for a message, the message transmission is complete and the chunks can be put together.
   In order to archive this, a ByteArray is allocated first. To be able to write into the array, it is wrapped in a ByteBuffer.
   After the chunks have been added to the ByteArray, the proxy checks if its hash equals the hash of the transmitted message. If this is the case, the message is deserialized by Kryo and transmitted to the receiver actor.

r, WMS 10MB



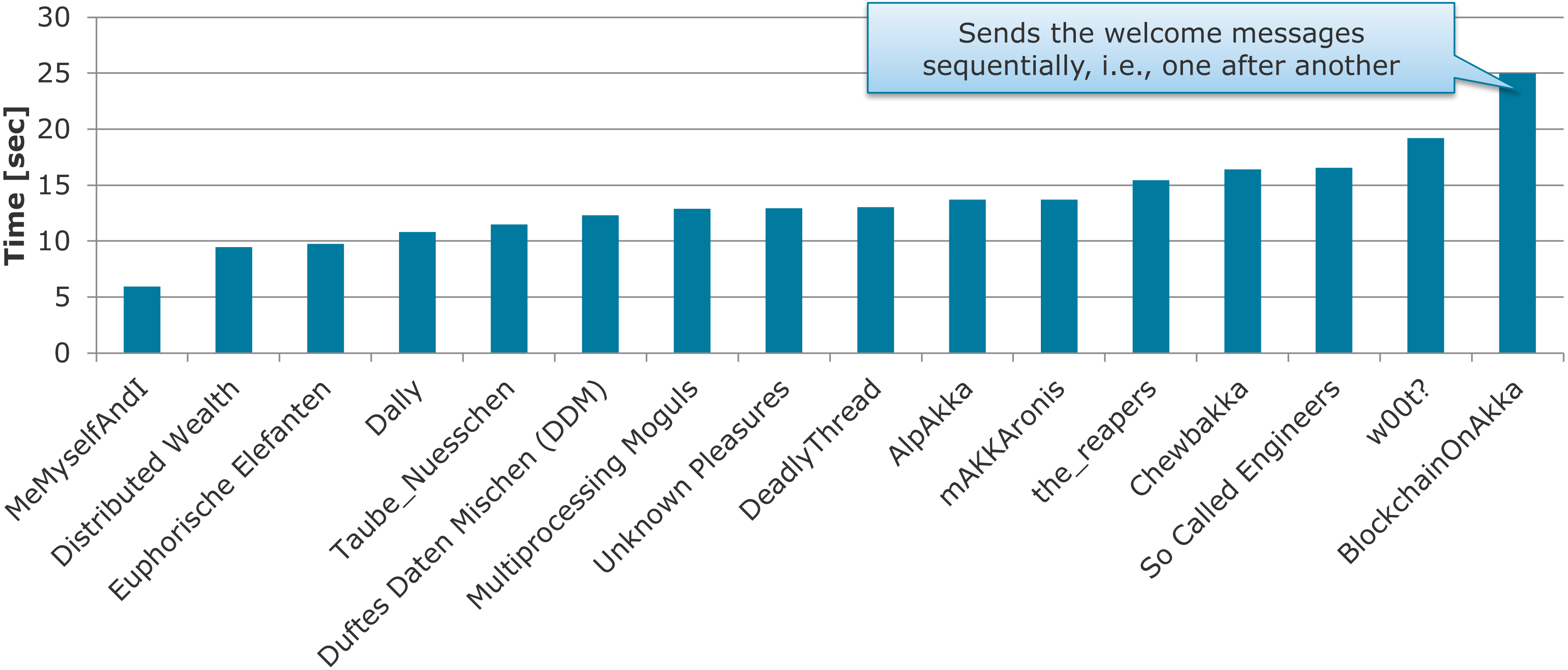
Assignment 2

# 1 master, **11** worker à 1 worker, WMS 10MB



# Assignment 2

## 1 master, **11** worker à **10** worker, WMS 10MB

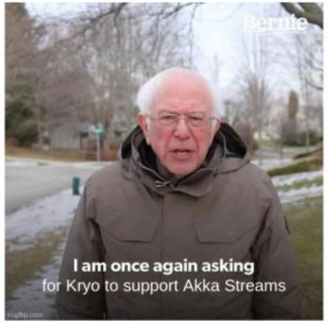


# Team BlockchainOnAkka: LargeMessageProxy

We use Akka Streams.


sending data over the network

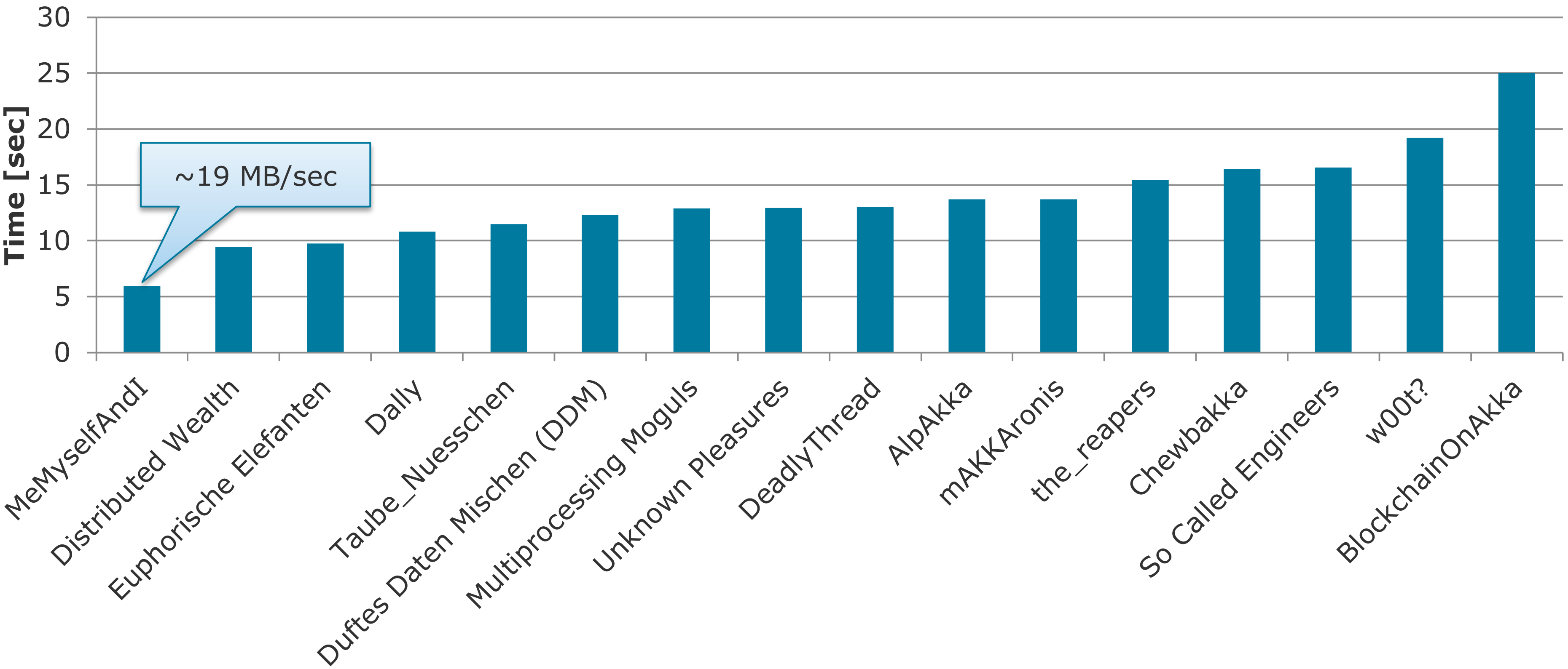
is faster than compressing it




Sender
LargeMessage<T>
-originalMessage:: T
-receiver: ActorRef
LMP (A)
Kryo Serialization of T
MessageOffer
-sourceRef: SourceRef<ByteString>
-sender: ActorRef
-receiver: ActorRef
feed ByteString(s)
AkkaStream of compressed ByteString(s)
pull ByteString(s)
LMP (B)
Kryo Deserialization of (merged) ByteStrings to Message T
T
From: Sender
Receiver

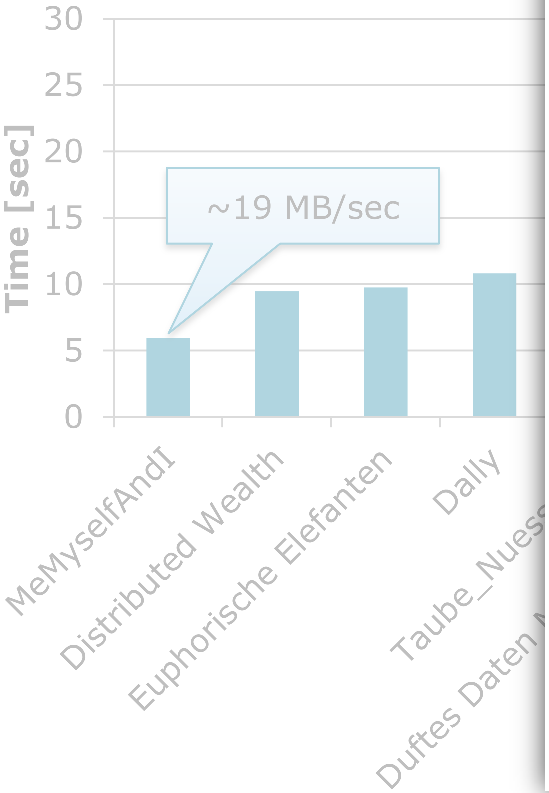

Assignment 2

# 1 master, **11** worker à **10** worker, WMS 10MB



Assignment 2

1 master, 11 w




# Large Message Proxy

## State

| | |
|---|---|
| messages being expected | Dict < messageID : metadata > |
| messages being received | Dict < sender : bytes received so far > |
| messages waiting to be sent | Dict < messageID : message > |

## Behavior

Sender
- Gets message
- message -> messages waiting to be sent
- sends request to send with ID and metadata (with timeout)

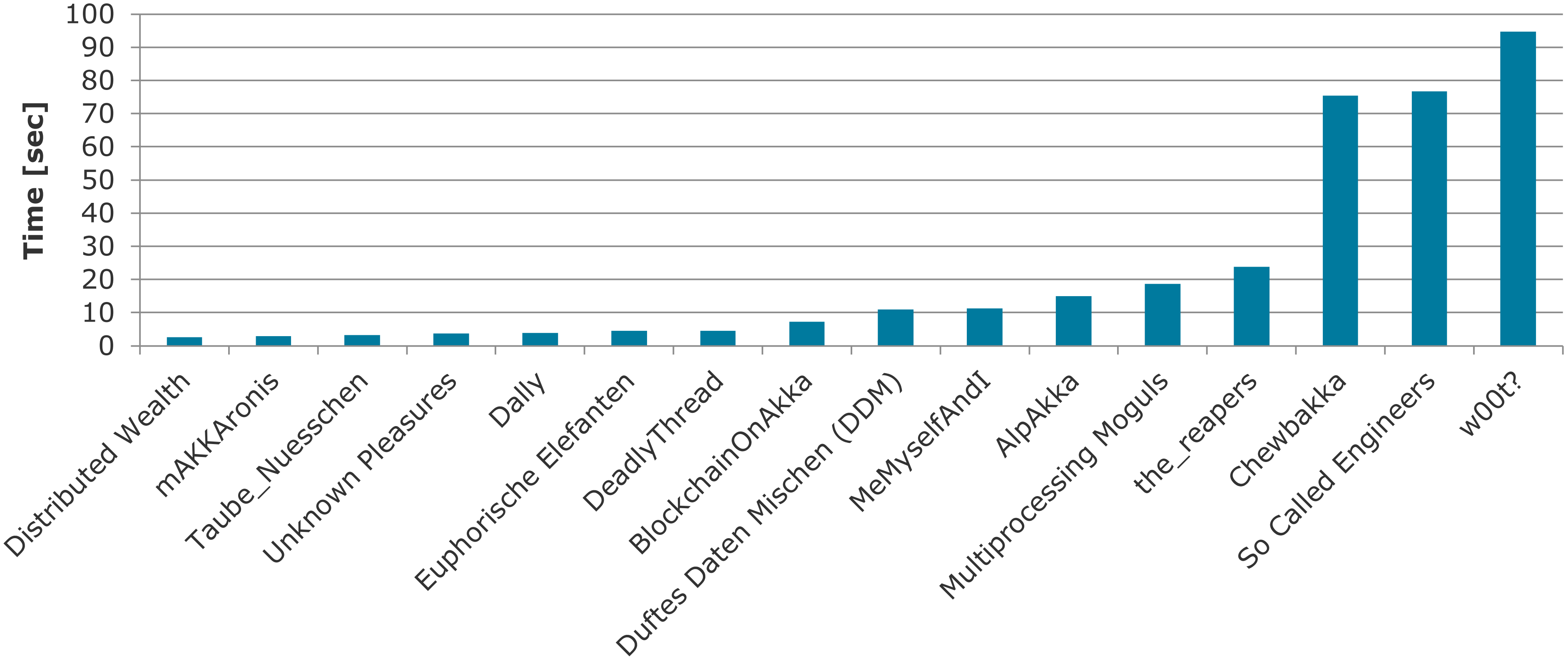
Receiver
- ID and metadata -> messages being expected
- sends ACK with ID

Sender
- if timeout:
  - resend request
- else:
  - delete ID from messages waiting to be sent
  - start streaming serialised( message with ID )

Receiver:
- adds stream packages to respective entry in messages being received
- on stream completion reassembles metadata and message by ID
- sends message to parent node

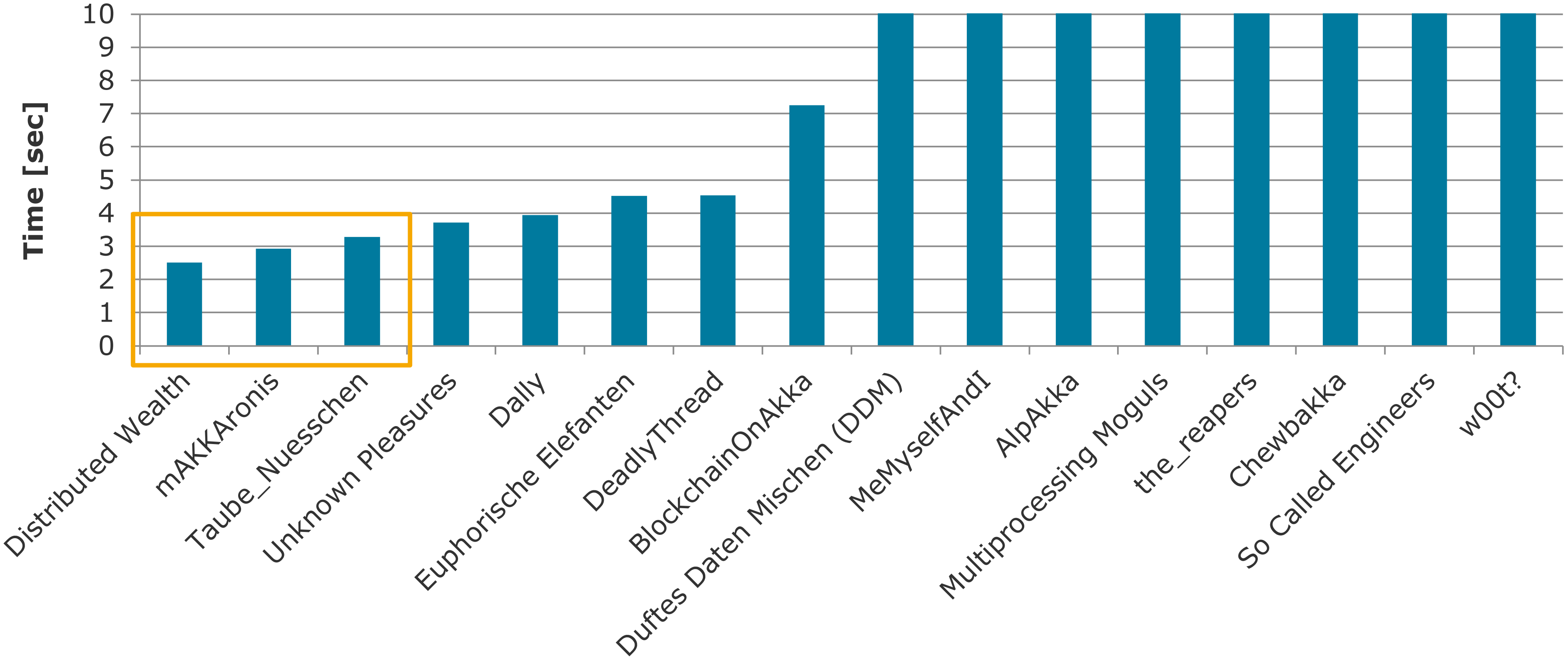
Assignment 2

# 1 master, 1 worker à 1 worker, WMS **100**MB



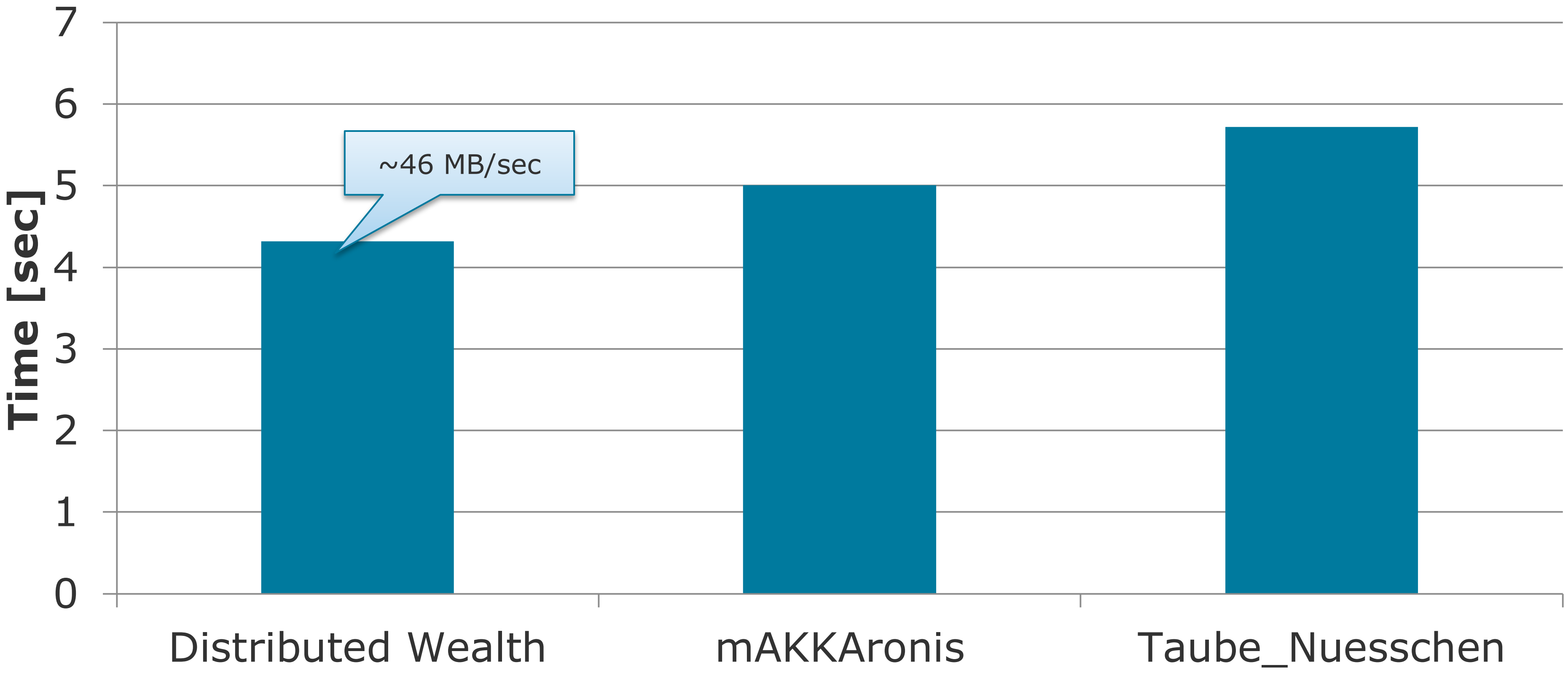
Assignment 2

# 1 master, 1 worker à 1 worker, WMS **100**MB

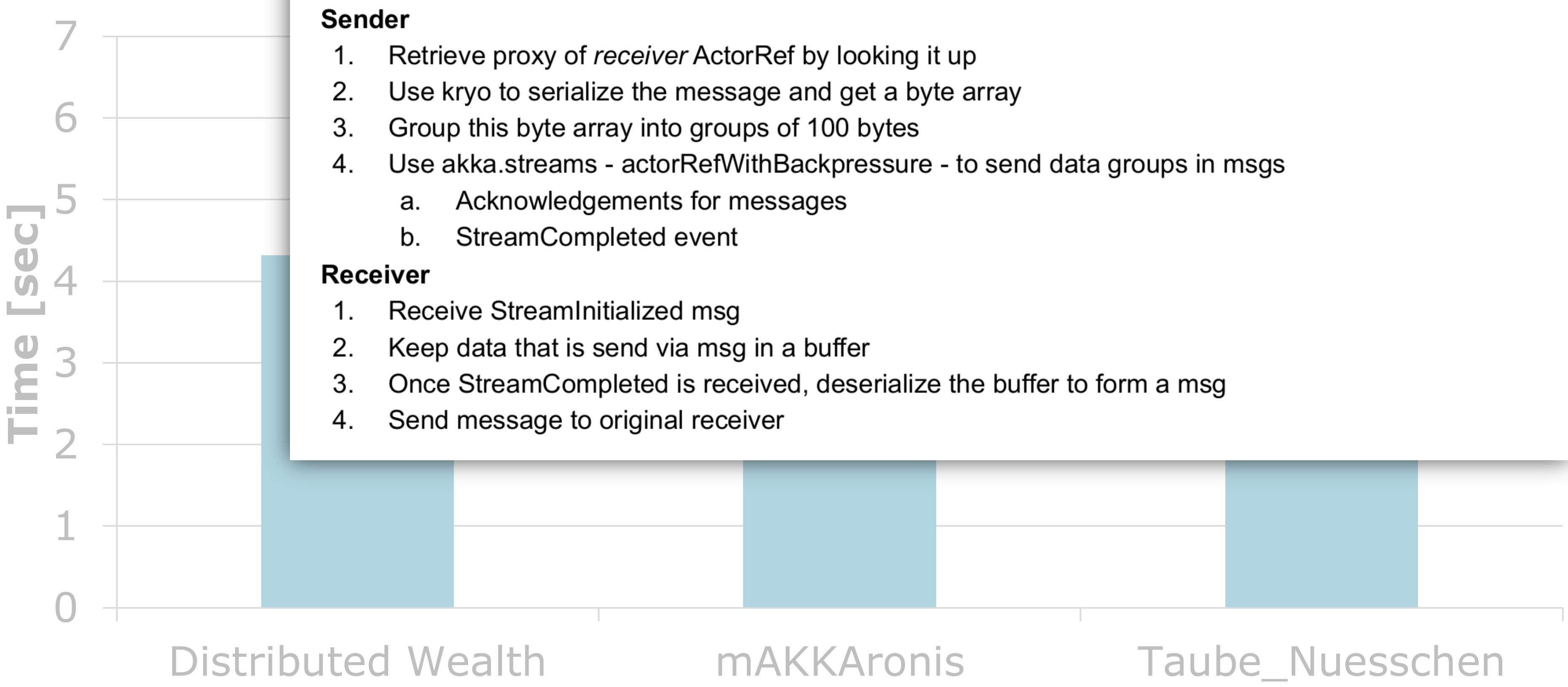


Assignment 2

# 1 master, 1 worker à 1 worker, WMS **200**MB



Assignment 2
1 master,




# Assignment 2

**Sender**

1. Retrieve proxy of *receiver* ActorRef by looking it up
2. Use kryo to serialize the message and get a byte array
3. Group this byte array into groups of 100 bytes
4. Use akka.streams - actorRefWithBackpressure - to send data groups in msgs
   a. Acknowledgements for messages
   b. StreamCompleted event

**Receiver**

1. Receive StreamInitialized msg
2. Keep data that is send via msg in a buffer
3. Once StreamCompleted is received, deserialize the buffer to form a msg
4. Send message to original receiver

# Assignment 2

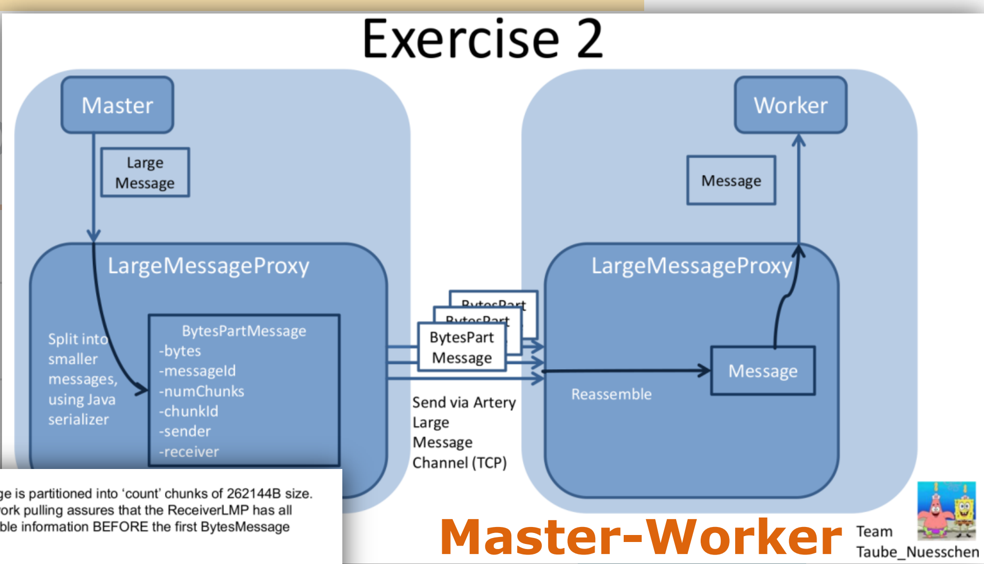
**Sender**

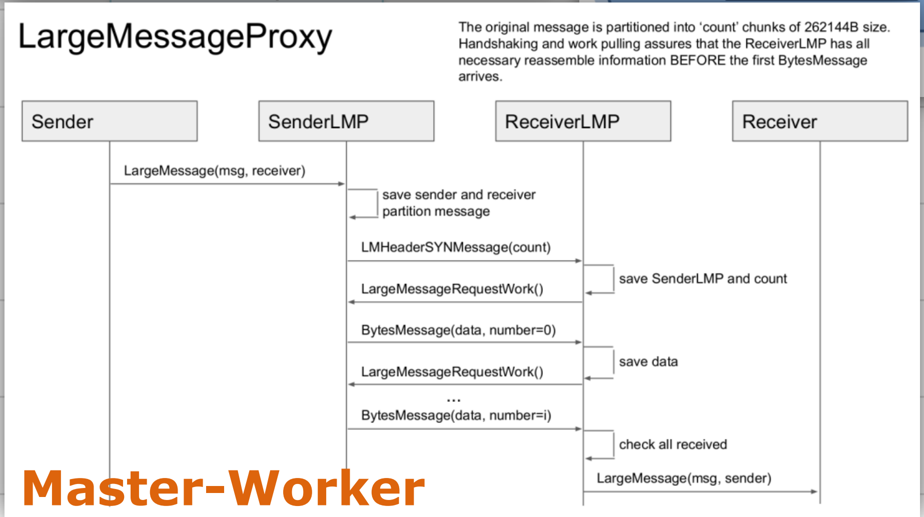
1. Retrieve proxy of *receiver* ActorRef by looking it up
2. Use kryo to serialize the message and get a byte array
3. Group this byte array into groups of 100 bytes
4. Use akka.streams - actorRefWithBackpressure - to send data groups in msgs
   a. Acknowledgements for messages
   b. StreamCompleted event

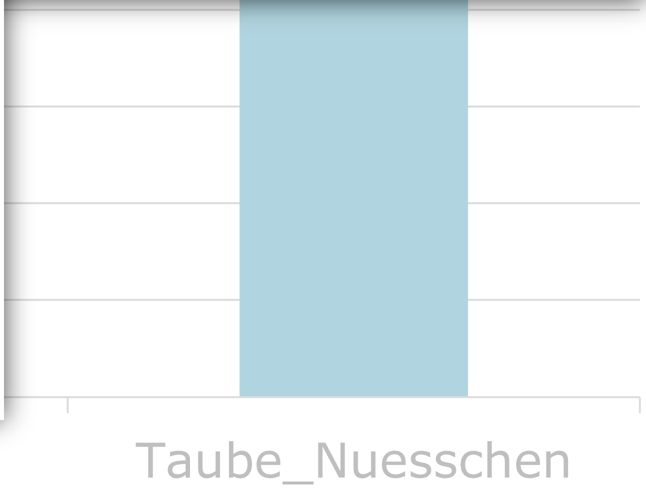
**Receiver**

1. Receive StreamInitialized msg
2. Keep data that is send via msg in a buffer
3. Once StreamCompleted is received, deserialize the buffer to form a msg
4. Send message to original receiver

**Akka Streams**

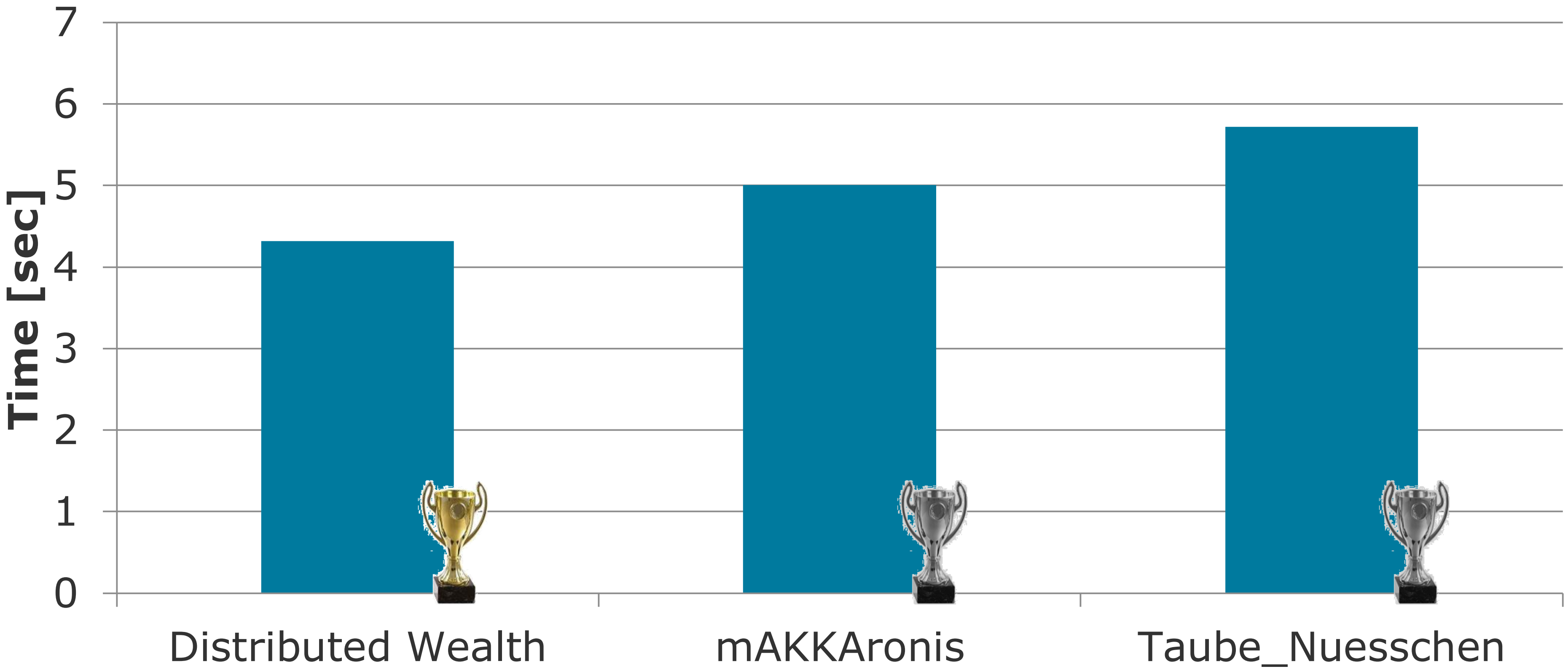


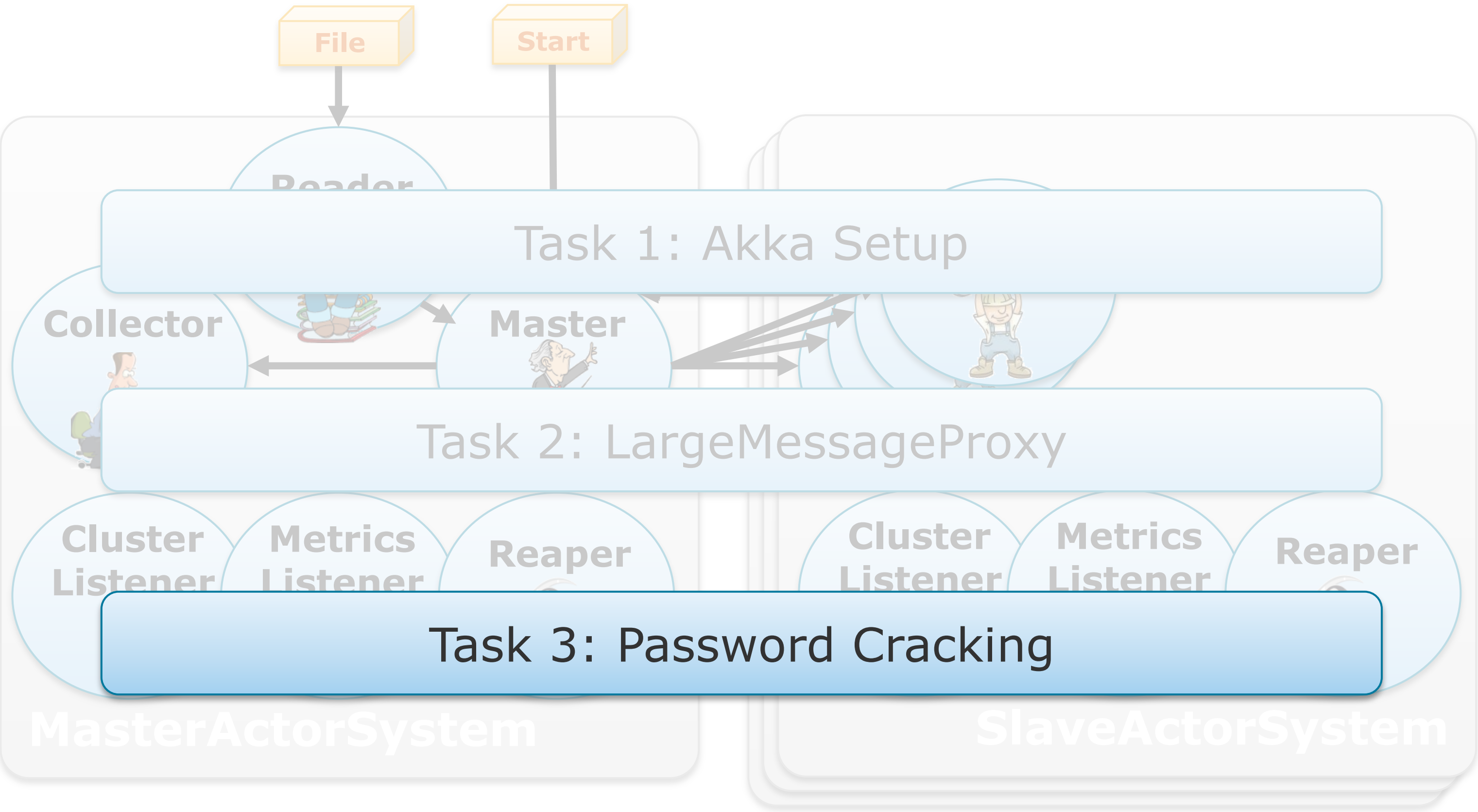
Assignment 2

# 1 master, 1 worker à 1 worker, WMS **200**MB



Homework

# Tasks / Assignments



Homework

# Task 3 – Password Cracking

| ID | Name | PasswordChars | PasswordLength | Password | Hint1 | Hint2 | Hint3 | Hint4 | Hint5 | Hint6 | Hint7 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Sophia | ABCDEFGHIJK | 10 | GGGFGFFFFG | HJKGDEFBIC | FCJADEKGHI | FAJBDIEKGH | AGCJEHFKIB | BHKICGFADJ | JIFAGKDBCE | GAHDKJBCEF |
| 2 | Jackson | ABCDEFGHIJK | 10 | EFFF[illegible] | [illegible] | AEHJIDGFKC | IDAHFGEKBJ | EHFIJKBGAC | HFJIEDACBK | FGKIDJCEAB | KDHGCAEJFB |
| 3 | Olivia | ABCDEFGHIJK | 10 | KDDD[illegible] | [illegible] | CAKEIFHGJD | JBFEDHIKAG | IDAKGHBFJC | KGBAEICHDJ | DKHFBEJIAC | EABJGFIKDC |
| 4 | Liam | ABCDEFGHIJK | 10 | CCCCGGCC[illegible] | CFHDBJKGEI | FAICGJDHEK | CHBKIGEJAF | AICDKGHJBF | EDAGKBJHIC | JDKIFACEGB | BGKJDAHCFE |
| 5 | Emma | ABCDEFGHIJK | 10 | BDDBDDDDDB | EGICDFKHBJ | HEAJIBDGFK | BAHCKDFIJG | HBEDKAGCIJ | IBHCEFJADK | FAGDEJICKB | GFHEAKCDBJ |
| 6 | Noah | ABCDEFGHIJK | 10 | GHGGHGGHHH | CFKBJGDIEH | CAIGHEJFDK | GJBEKIADFH | AIBCJHGEKF | GDIBCKFHAJ | CGJHDEAIBK | DGKFBEACJH |
| 7 | Ava | ABCDEFGHIJK | 10 | DFFFDFDFDD | FHIKEBGDJC | KHFICAJGED | KIAHDFEJGB | CGFAKIBDHJ | ACEHFKBIDJ | GBADIJEKFC | AFCKGHBDJE |
| 8 | Aiden | [illegible] | [illegible] | [illegible]HHIHI | GCIFEHDKBJ | JDHIEGKACF | FJHBEGAKDI | AIBJEHKGFC | CGJAFBIHDK | JECAIDGHBK | IJBDCKEAFH |
| 9 | Isabella | [illegible] | [illegible] | [illegible]CJCJCCJ | EHDCGIKBJF | IFJCAEHGKD | AFBHEGKIJC | KGFBIADJHC | JKAGEDHIBC | CBKIDEAHFJ | CJAFKEIBDG |
| 10 | Lucas | [illegible] | [illegible] | [illegible]BBCBCCC | KGJHIDECFB | BHFACKEGIJ | ICJGHFKBAD | KEICHGAJDB | BCDKEJIFAH | IDGEBAJKCF | FCBDKGHJAE |
| 11 | Mia | AB[illegible] | 10 | IDDDDDIDDI | FIB[illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | BKIGAEDFCJ |
| 12 | Caden | ABCD[illegible]IJK | 10 | DDDAADDDDD | AEI[illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | EKFGAJCBHD |
| 13 | Aria | ABCDEFGHIJK | 10 | CCCFCCFFCC | GKH[illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | HBJEAFDCGK |
| 14 | Grayso[illegible] | [illegible] | [illegible] | BJBJJJBBJJ | GEH[illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | BKGEJFACDI |
| 15 | Riley | [illegible] | [illegible] | [illegible]BGGGBBB | GHE[illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | JBCHAGEKDF |
| 16 | Mason | [illegible] | [illegible] | [illegible]AAJAJA | EKJ[illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | BCGEFAKJID |
| 17 | Zoe | [illegible] | [illegible] | [illegible]AJJJJJJ | JHC[illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | CBEFIDGKAJ |
| 18 | Elijah | ABCDEFGHIJK | 10 | EJJEJEJJEE | KDC[illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | JBCGADIEKF |
| 19 | Amelia | ABCDEFGHIJK | 10 | GDDGGGDGDG | GCI[illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | HEGDCAKJFB |
| 20 | Logan | ABCDEFGHIJK | 10 | FFEFEEEFFF | KHB[illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | JFHDCKEAGB |
| 21 | Layla | ABCDEFGHIJK | 10 | CCCHCHCCCC | GIF[illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | JBGAHFDCEK |
| 22 | Oliver | ABCDEFGHIJK | 10 | ABBBAABAAA | AFK[illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | FJGKHEADBC |
| 23 | Charlotte | ABCDEFGHIJK | 10 | BGBGBGBBBG | ECK[illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | GJBECKAFHD |
| 24 | Ethan | [illegible] | [illegible] | [illegible]HHBBHHH | DHJ[illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | BCHJEAKFDG |
| 25 | Aubrey | [illegible] | [illegible] | [illegible]EJJEEJE | HDFEJBKICG | IF[illegible] | [illegible]HGEBIK | KCEFBAIGHJ | IDCJBAGEHK | IKHJCDBAEF | JCGIFDBEAK |
| 26 | Jayden | [illegible] | [illegible] | [illegible]CCCGGGG | DJEHCBKFIC | [illegible]JAIDHE | ECFKBIAHJG | GDCIFKBJAH | HJBGAIKCED | DICKFBGAEJ | GFDCKBAHJE |
| 27 | Lily | [illegible] | [illegible] | [illegible]DHHDHDD | KCJFIBHEDG | FJDKCAIEGH | DABGJEFKIH | DCGIKHFABJ | KDGBEHIACJ | IBCKDFHJAE | EHABDKCFJG |
| 28 | Muham[illegible] | [illegible] | [illegible] | [illegible]CBCBBBC | EDFGHKIBJC | HEICAKBJFG | CABIDFGHKJ | DHAKICBGJE | IHAKJCEBFD | CEABFJGKID | ABGKFDHCEJ |
| 29 | Chloe | ABCDEFGHIJK | 10 | CEECECCCEC | KEGDHFCBIJ | GCIHAEDKFJ | HFGKIBACEJ | CJHGKBDAIE | FECIBJKADH | GAFCIBEKJD | CJBAKEGDFH |
| 30 | Carter | ABCDEFGHIJK | 10 | BBIIBIBIIB | CKFGBIHDEJ | EJDKIHGABF | ECHIJGFBAK | AFHIBCKGDJ | IHCBKGEJAD | AFHIDJKBCE | IBGCJKFAED |
| 31 | Harper | ABCDEFGHIJK | 10 | EAAAEAEEAE | IAFKCHJGDE | AKFDJIHGBE | BGECFIJKAH | BJDAIGKEHC | IHEBKACJDF | BJDIGAKFCE | EFADJKCBGH |
| 32 | Michael | ABCDEFGHIJK | 10 | DCDDCCDDDD | JGFEICBKHD | CKJDHGIEAF | KIGDHABJCF | GDEIHACBJK | BICEAFDHKJ | JGFCKBDEAI | HCBGDKFAJE |
| 33 | Evelyn | ABCDEFGHIJK | 10 | IICICCICCC | CDBJHIEGKF | FIKGEHCAJD | BIJAGEKFHC | CADBIHFJKG | GHJDBKAEIC | KJEIDHABCF | ACBGKFDIJE |
| 34 | Sebastian | ABCDEFGHIJK | 10 | EIIIEIIIIE | EDFKHGJBIC | KFHIDJGACE | KHAIDGBJFE | CIABGJKEHF | JKEIDCBAHG | KIDFJABHEC | FDBGJCAEIK |
| 35 | Adalyn | ABCDEFGHIJK | 10 | JJJJDJDDJJ | IKEJFCBHDG | AHDCFKEJGI | DBIKAHJGEF | DKAIHGFJCB | CABKHEDJIG | ABJEFCHDKI | GJAFEBCDIK |
| 36 | Alexander | ABCDEFGHIJK | 10 | HKHKHKKHKK | CEBHKDFIJG | FGJADKCIEH | AGDIFHKJBE | FHCEKABGIJ | BFJAIDCGHK | BCAHEJDGIK | JFCDIHKAEB |

Passwords to be cracked

All characters that may appear in the password

Number of characters in the password

These two fields have always the same value for all records.

Hints:

- Every hint contains all PasswordChars besides one char, i.e., |Hint|=|PasswordChars|-1
- The missing char is the hint, because it does not appear in the password.
- The number of hints can change!
- The more hints we have, the easier it is to find the password.

Homework

# Task 3 – Password Cracking

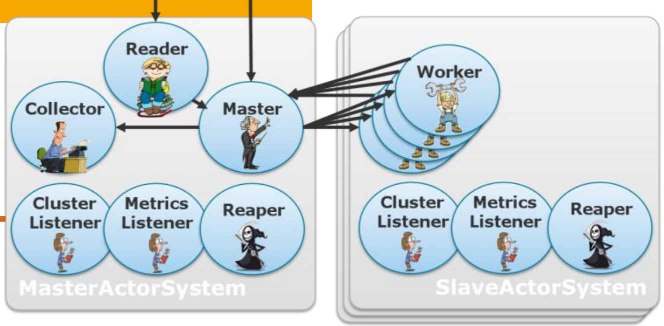
Both password and hints are SHA-256 encrypted.

Encryption cracking via brute force approach:
1. Generate sequence.
2. Encrypt sequence with SHA-256.
3. Compare current SHA-256 with existing one: if equal, encryption is broken.

Hint cracking is much easier than password cracking.

| ID | Name | PasswordChars | PasswordLength | Password | Hint1 | Hint2 | Hint3 | Hint4 | Hint5 | Hint6 | Hint7 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Sophia | ABCDEFGHIJK | 10 | c4712866799881ac48ca55bf78a9540 | 1582824a01c4b | e91aca467f5a2 | 52be0093f91b9 | 8052d9420a20c | ca70f765d8c1b | 570d3ada41de | f224061bd0359 |
| 2 | Jackson | ABCDEFGHIJK | 10 | c178ef3bd2dbf4e92291a9b563c0ae2c | 7624e76e72b52 | 834d255d0276 | b2e939a89b78 | 0f0c2aefcfcf4b3 | d22b58963201 | 0066eb98a0f3 | 21b5a6f0b9c15 |
| 3 | Olivia | ABCDEFGHIJK | 10 | b6d | | | | | 0dd9e2605994 | f70853fec1c11 | b0f110e28c9d |
| 4 | Liam | ABCDEFGHIJK | 10 | 109 | | | | | 90d6920945a7 | b857b99db7ab | 503d34487226 |
| 5 | Emma | ABCDEFGHIJK | 10 | 607 | | | | | 552ba27c5ae4 | 60b64d370b66 | b7fdd9f77b932 |
| 6 | Noah | ABCDEFGHIJK | 10 | 6d4 | | | | | a1601cb73654 | 62ecbbd80652 | a1ba7bb71eb9 |
| 7 | Ava | ABCDEFGHIJK | 10 | 4121ab0055971 | e00595b2cab3c | 90d6247cbef52 | 72c52457352b | 7ff4495040445 | 628dfdd46cd2f | c901b559232e | b8209fa626310 |
| 8 | Aiden | ABCDEFGHIJK | 10 | fbe3613750f71d7996e9d63601dc7fd4 | de2617fb757fc6 | 06bb6d175e5d | 03ee78244a72 | 87316b71fbfc44 | 9aab84d04556 | 87a65ceb83b5 | 589c35f40243e |
| 9 | Isabella | ABCDEFGHIJK | 10 | 5a22e3bdef6c85307b361f2e1758f461 | 23d6de9da425b | 7af3c5c070a12 | 824137665f56 | c71deb0e1e18a | 49535ddb45d7a | 9271a854a0e6 | 6b2b2dbf84e0a |
| 10 | Lucas | ABCDEFGHIJK | 10 | 49afdd0a20ae497060405ec7b557faa6 | 041734164643 | 0df7feecbe4bb | 046c1ffec90e0 | a221a7c41ebf4 | dcbe04357c159 | ae51984b3c8c | e5b090db2396 |
| 11 | Mia | ABCDEFGHIJK | 10 | 77026d73fb8c33e0f45c3f6bc3 | | | | | | 30469e377b | 5b451e4478c4 |
| 12 | Caden | ABCDEFGHIJK | 10 | 484616315092a69ebd7cf4c1b | | | | | | 5570b879d43 | eb83b37c3c50 |
| 13 | Aria | ABCDEFGHIJK | 10 | 3fff9b667a867fccaada0d823d0 | | | | | | d3bbb35405 | 39b7290d29bd |
| 14 | Grayson | ABCDEFGHIJK | 10 | ac923aa891c087fad57b02de9 | | | | | | bb614d9f1c9 | 485a4dbf7cc98 |
| 15 | Riley | ABCDEFGHIJK | 10 | 57203d2db503c69464900aed | | | | | | a1595ef921e | 6350339f168b8 |
| 16 | Mason | ABCDEFGHIJK | 10 | 4d873360dd931098ead7d692 | | | | | | 7433be73e8a | 7aa536698df56 |
| 17 | Zoe | ABCDEFGHIJK | 10 | f2095d3f48f6c0366423436865 | | | | | | 023d0fbf0d4a | f9f4dfc2082c69 |
| 18 | Elijah | ABCDEFGHIJK | 10 | 25e975a018dd7265dcb44a17 | | | | | | e3d813569ce | 698303662587 |
| 19 | Amelia | ABCDEFGHIJK | 10 | 6fb693ee39e015290f087a0ca | | | | | | e9182a060c6 | 584a163c9c80 |
| 20 | Logan | ABCDEFGHIJK | 10 | 1d43da0376f725fff867e1096e3635c9a | 8fb | 9faf4b0945b | 41c672365d64 | dfcd28d1604f0a | 45759a68fbe9c | 1c8b4ed5fccc3 | b58d913c2aac |
| 21 | Layla | ABCDEFGHIJK | 10 | c3647d6d4f8e8136cf7640d1976d2346 | 78b25e3 | 494879c4612ff | 70d759f30b22c | 74e53e2720fee | 10a55cb79be0e | 56459103a26b | efaee8c12e011 |
| 22 | Oliver | ABCDEFGHIJK | 10 | d2488287e89e2bb00bff6c4e767fe5 | c4c588e7b790b | 911eca14216d | f52b85e1bb8c3 | 3ba17b3532e81 | 8848a569dcd01 | f9b2ecef6af24 | 9ab4b25771e3 |
| 23 | Charlotte | ABCDEFGHIJK | 10 | 0e481c55eea1567bf4a5434cc0d713d | d6426c5a36fa4 | 7578c180f1b19 | 46588a7ef05e0 | 5daadc4464d5c | bf1244e8e8879 | fe47bd94da1e | 7a8c1b2e2827 |
| 24 | Ethan | ABCDEFGHIJK | 10 | d08ce9b35434a29b6d34ae4df99114e | 537ceb64562eb | ac4c0b2db991 | b43e80f0be336 | cccd5b0f386b8 | 868e118ee7dd | eed78dc7c439 | e30e4a6278bf |
| 25 | Aubrey | ABCDEFGHIJK | 10 | a54 | | | | | dbef55053480e | 39769b86753a | ec515cf86d999 |
| 26 | Jayden | ABCDEFGHIJK | 10 | 482 | | | | | 0b28c5a0bdae | d43e7731d10e | 09c6383d87f59 |
| 27 | Lily | ABCDEFGHIJK | 10 | 64e | | | | | 71545ceb163c0 | 0b5a12e03c63 | 36b071418c49 |
| 28 | Muhammad | ABCDEFGHIJK | 10 | b24 | | | | | c72f7ec4768b1 | ff8ffabe05c576 | 73170f8660b2b |
| 29 | Chloe | ABCDEFGHIJK | 10 | 314885f3b250cfad9a08ab7c6a0b7125 | ba60bc240c6f8 | b32fc6c704 | | e9d2d120af79 | e2a2fb62382d6 | 1e52d3a60770 | 9e6240b987eb |
| 30 | Carter | ABCDEFGHIJK | 10 | 507b389927e0aa92bdf50e7ffe0c119c | 221935370639 | ec62e5d714ffd | f4ce80b6 | 3e0e6e85ce | 7c1163a66461 | 18e32024bb20 | 4d35243c99da |
| 31 | Harper | ABCDEFGHIJK | 10 | 17649029a718c93179e9da331e78012 | f4aa95f0083c66 | b0c11abac12a | 98878f2bea2d9 | 61101eb1bfae6 | d694ff1668eca5 | 2a2e2681f0d0 | 5194a8888927 |
| 32 | Michael | ABCDEFGHIJK | 10 | a926deae7e334a3992fbfa30d4d75823 | 8b63be6310da5 | 03fbdc4f9b69fa | d96188307f70 | 6bdcb76976d88 | e54cb6eabaa20 | 9c2f8383f1aa5 | 04bd3a11b5e4 |
| 33 | Evelyn | ABCDEFGHIJK | 10 | 43079487b664ebafba46e77698d58a4 | 7b43a0546a75f | c6c9a5d45cf19 | 67cbc51d481e | b1a79b2429508 | 59099a87582e | ac6479c44e48 | 7d490919fceb7 |
| 34 | Sebastian | ABCDEFGHIJK | 10 | 0306aed6a72de9d32e0b9d9ec430e92 | 8e5837886ae8a | 2a9f2b7b2e974 | 5effda9ae8fd9 | 037bcf1d83a00 | 3d4b9e8ba7bca | 9379b2c12b91 | f1f3b79be05dc |
| 35 | Adalyn | ABCDEFGHIJK | 10 | bef1a0cc6ba9868fe2071e80b7069f24 | 622ba2b0c4557 | 1087ebc69f50b | c28553d4a058 | 74802c5978ebf | 50e770146296 | ba4c14f03ca99 | d2bda4956404 |
| 36 | Alexander | ABCDEFGHIJK | 10 | f14a798017874d94e78421db5a126e6 | 50e4f0b88e214 | b5502b12a7d7 | d897e5993c0d | 11547ce885e7c | aa8e5f28e181e | cd8b68b0cdba | da8408b0a088 |

# Task 3 – Password Cracking




## Hints

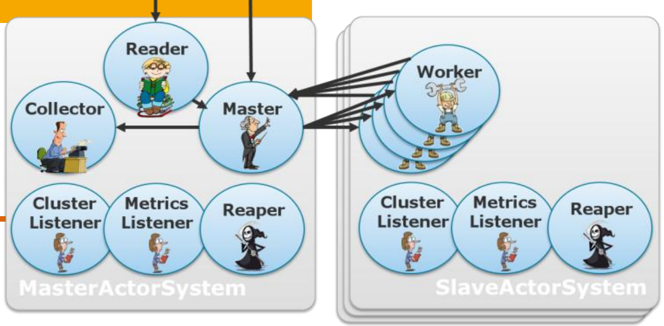
- The passwords and hints are encrypted with the following function:

```
private String hash(String password) {
   MessageDigest digest = MessageDigest.getInstance("SHA-256");
   byte[] hashedBytes = digest.digest(line.getBytes("UTF-8"));
   StringBuffer stringBuffer = new StringBuffer();
   for (int i = 0; i < hashedBytes.length; i++)
      stringBuffer.append(Integer.toString((hashedBytes[i] & 0xff) + 0x100, 16).substring(1));
   return stringBuffer.toString();
}
```

- Useful code snippets for combination generation:
  - https://www.geeksforgeeks.org/print-all-combinations-of-given-length/
  - https://www.geeksforgeeks.org/heaps-algorithm-for-generating-permutations/

Homework

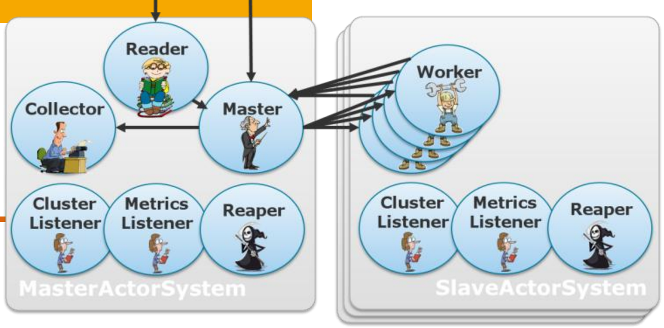
# Task 3 – Password Cracking




Hints

- Think agile:
  - How can I maximize the parallelization?
    (e.g. the number parallel tasks should in the best case not depend on the input data)
  - How can I propagate intermediate results to other actors whenever needed?
    (e.g. proxies, schedulers, master-worker, …)
  - How can I re-use intermediate results to dynamically prune tasks?
    (e.g. if I know that X is a solution, then I might be able to infer without testing that Y is also a solution)
  - How can I implement task parallelism?
    (e.g. parts of subtask 2 might already be able to start with partial results of subtask 1)
  - How can I achieve elasticity in the number of cluster nodes?
    (nodes may join or leave the cluster at runtime)

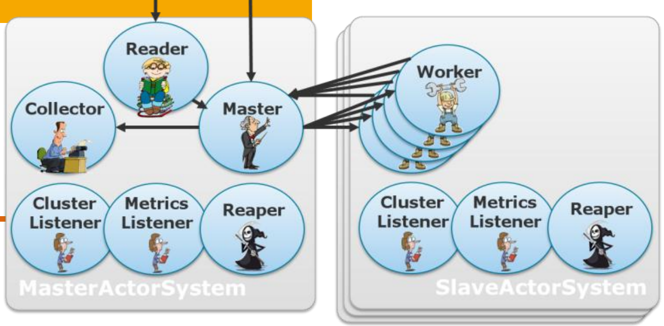
# Task 3 – Password Cracking




Notes

- Parameters that may change:
  - password length
  - password chars
  - number of hints (= width of file)
  - number of passwords (= length of file)
  - number of cluster nodes
    (do not wait for x nodes to join the cluster; you do not know their number; implement elasticity, i.e., allow joining nodes at runtime)
- Parameters that may not change:
  - encryption function SHA-256
  - all passwords are of same length and have same character universe

# Task 3 – Password Cracking




## Rules

- Do not mess with the time measurement:
  It should start with the StartMessage and it should end when the PoisonPills are sent.
- Do not change the command line interface or app name;
  otherwise, the automatic test scripts will fail.
- Use maven to import additional libraries if you need some.
- Do not use the disk.
- Feel free to change everything (besides interface and time measurement);
  you probably need a new shutdown protocol, you need a proper communication protocol for your Master/Worker actors and you probably need additional actors.
- Write the cracked passwords with the Collector to the console;
  the current printouts from the master should be deleted.

Assignment 3
# Solution Approaches

For both, start cracking the overall password as early as possible:
- as soon as all its hints are cracked
- or even earlier by guessing the password letters

### Approach 1: "Straight-forward cracking"

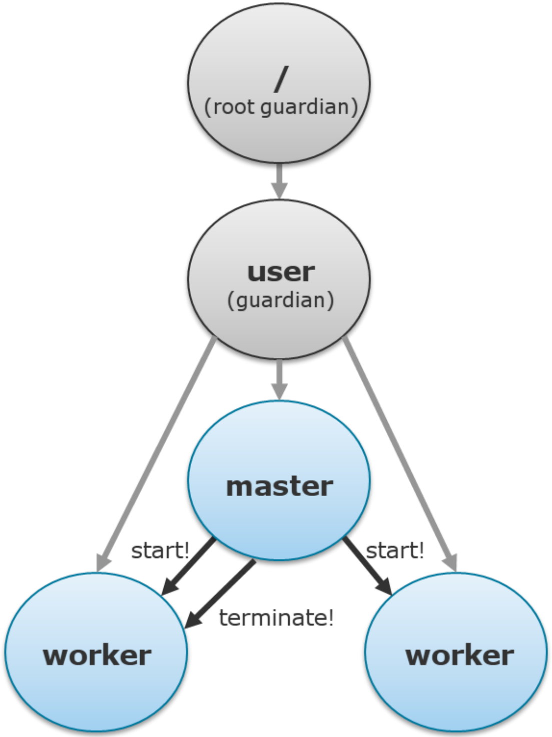
- 1 user = 1 task
- Partition passwords by users.
- Distribute all users and crack the passwords in parallel.
- Crack the hints first, then crack the password.
  - Optimization: Crack the hints in parallel by spawning child actors.

### Approach 2: "No redundant hashing"

- 1 hint letter = 1 task
- Replicate the hints (and passwords) to all workers.
- Partition the hint space (e.g. 1 hint letter = 1 task).
- Each worker creates all hash-representations for its hint and checks which passwords use it.
  - Optimization: More fine-grained hint space partitioning, e.g., by using the hint letter as primary partitioning criterion and the letter permutation prefix as secondary partitioning criterion.

Assignment 3

# General Feedback




### "The non-reactive workers"

- If workers are tasked to crack many passwords, they are unresponsive for some time.
- This can lead to non clean shutdowns.
  - ➢ Keep tasks small and/or actively check inboxes once in a while.

### Connection-, Future- and Stream-Errors

- Let all Actors carefully close their resources before you terminate them!

### "The tedious-hashing workers"

- Idea: Create permutations/combinations on master and send hashing tasks.
  - ➢ Master needs to send too much data (network becomes the bottleneck)
  - ➢ Hashing tasks are too small (too much scheduling for too short tasks)

Homework

# Task 3 – Test

| Team | Executes? | Terminates? | Distributes? |
|---|---|---|---|
| supreme-broccoli | | | |
| w00t? | | | |
| Code Monkeys | | | |
| ddm_team_42 | | | |
| Duftes Daten Mischen (DDM) | | | |
| Dally | | | |
| Distributed Wealth | | | |
| the_reapers | | | |
| Chewbakka | | | |
| BlockchainOnAkka | | | |
| Unknown Pleasures | | | |
| Taube_Nuesschen | | | |
| So Called Engineers | | | |
| Alpha | | | |
| Euphorische Elefanten | | | |
| mAKKAronis | | | |
| MeMyselfAndI | | | |
| Multiprocessing Moguls | | | |
| AlpAkka | | | |
| DeadlyThread | | | |

# Homework

# Task 3 – Test

| Team | Executes? | Terminates? |
|---|---|---|
| supreme-broccoli | Yes | Yes |
| w00t? | Yes | Yes |
| Code Monkeys | Yes | Yes |
| ddm_team_42 | Yes | Yes |
| Duftes Daten Mischen (DDM) | Yes | Yes |
| Dally | Yes | Sometimes |
| Distributed Wealth | Yes | Yes |
| the_reapers | Yes | Yes |
| Chewbakka | Yes | Yes |
| BlockchainOnAkka | Yes | Yes |
| Unknown Pleasures | Yes | Yes |
| Taube_Nuesschen | Yes | Yes |
| So Called Engineers | Yes | Yes |
| Alpha | Yes | No |
| Euphorische Elefanten | Yes | Yes |
| mAKKAronis | Yes | Yes |
| MeMyselfAndI | Yes | No |
| Multiprocessing Moguls | Yes | Yes |
| AlpAkka | Yes | Yes |
| DeadlyThread | Yes | Yes |

**Observation:**
Sometimes action stops with no error message.
**Assumption:**
Algorithm parallelizes to a higher degree than provided number of workers; maybe that causes lost messages?

**KNOWN BUGS**

- workers sometimes don't solve for all hints after first password and therefore don't necessarily crack all passwords
- output of collector or timing for password cracking task sometimes doesn't appear

**Management**

Akka Actor

**Observations;**

- system stops without a result at some point: `akka://ddm/deadLetters`
- `akka://ddm/user/master/largeMessageProxy| null: java.lang.NullPointerException: null`

Homework

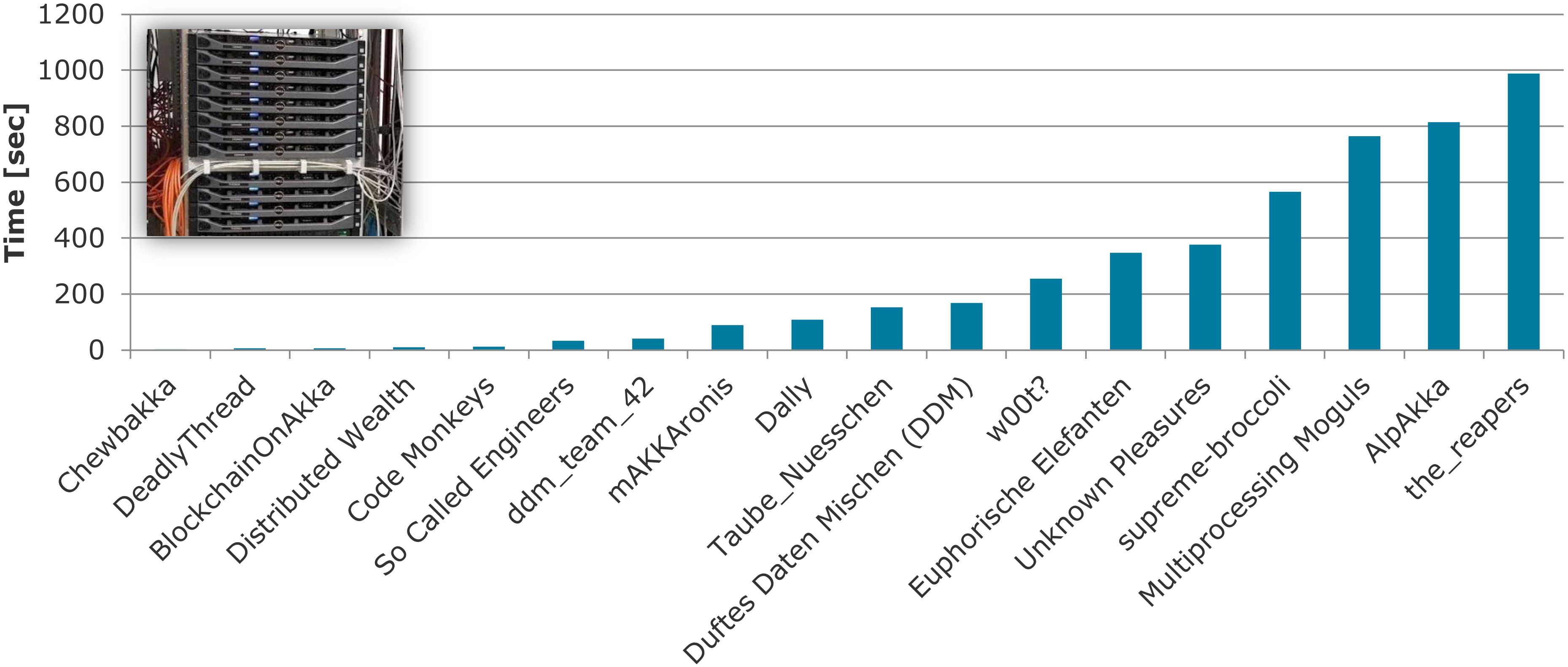
# Task 3 – Test

**Assumption:**
Still due to the Kryo vs. SourceRefImpl serialization issue.

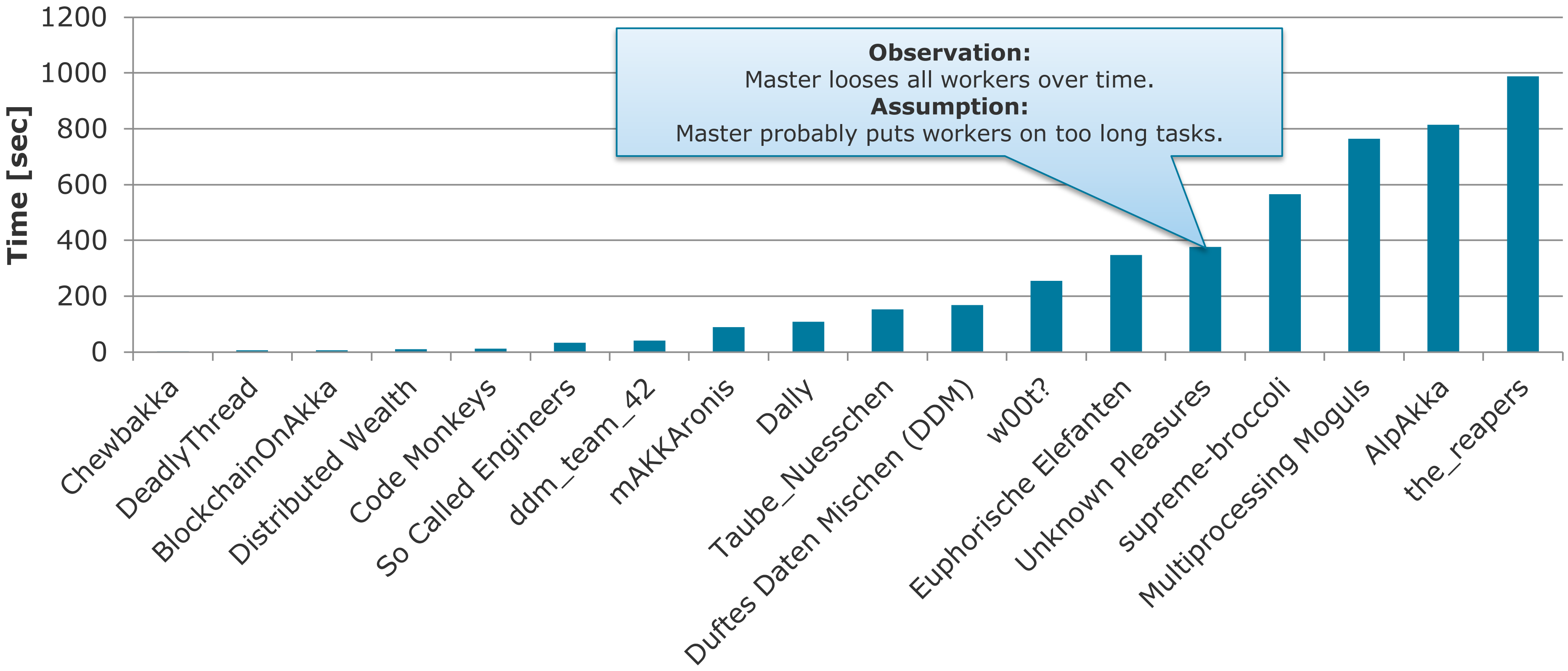
| Team | Executes? | Terminates? | Distributes? |
|---|---|---|---|
| supreme-broccoli | Yes | Yes | No |
| w00t? | Yes | Yes | Yes |
| Code Monkeys | Yes | Yes | Yes |
| ddm_team_42 | Yes | Yes | No |
| Duftes Daten Mischen (DDM) | Yes | Yes | Yes |
| Dally | Yes | Sometimes | [illegible] |
| Distributed Wealth | Yes | Yes | [illegible] |
| the_reapers | Yes | Yes | [illegible] |
| Chewbakka | Yes | Yes | [illegible] |
| BlockchainOnAkka | Yes | Yes | [illegible] |
| Unknown Pleasures | Yes | Yes | Yes |
| Taube_Nuesschen | Yes | Yes | Yes |
| So Called Engineers | Yes | Yes | Yes |
| Alpha | Yes | No | Yes |
| Euphorische Elefanten | Yes | Yes | Yes |
| mAKKAronis | Yes | Yes | Yes |
| MeMyselfAndI | Yes | No | Yes |
| Multiprocessing Moguls | Yes | Yes | Yes |
| AlpAkka | Yes | Yes | Yes |
| DeadlyThread | Yes | Yes | Yes |

**Observation:**
`akka://ddm/user/master/largeMessageProxy| null - java.lang.NullPointerException` prevents the worker nodes from helping the master node.

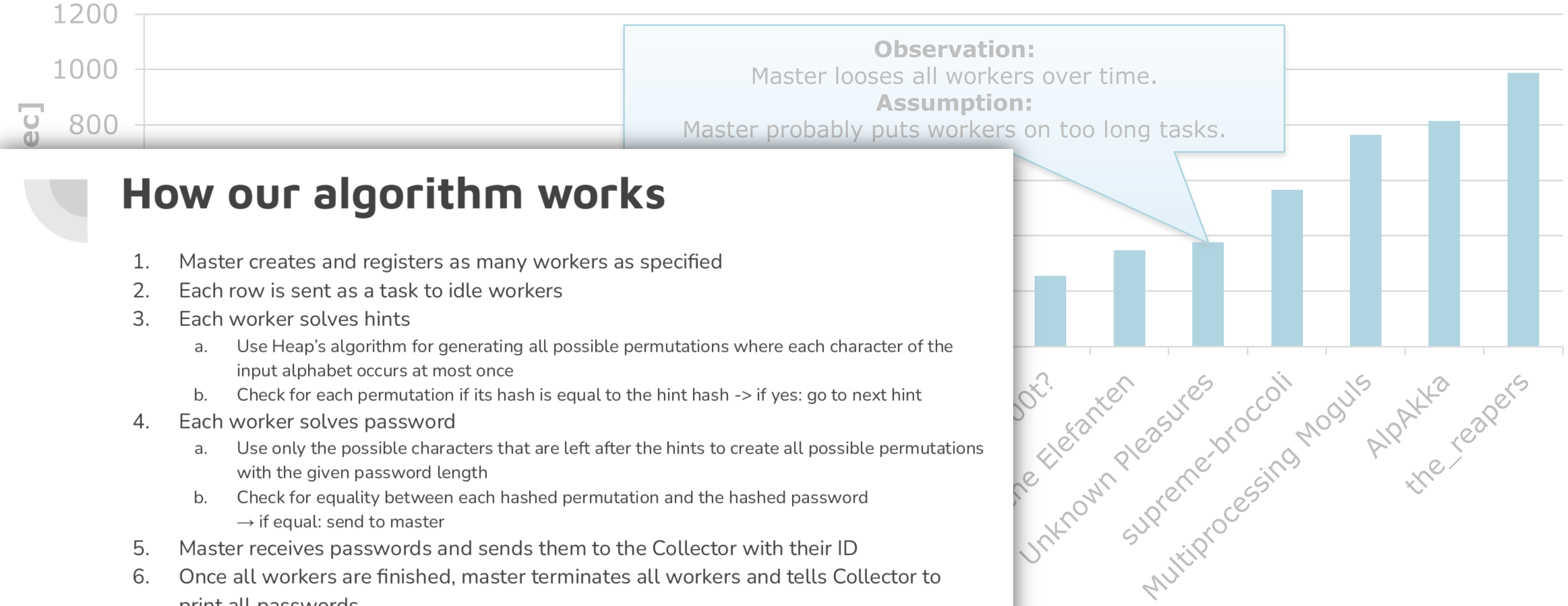
# Task 3 – students.csv
## 100 names; 10 length; 11 chars; 9 hints; 10 worker/node



# Task 3 – students.csv
## 100 names; 10 length; 11 chars; 9 hints; 10 worker/node



Task 3 – students.csv
100 names; 10 length; 11 chars; 9 hints; 10 worker/node

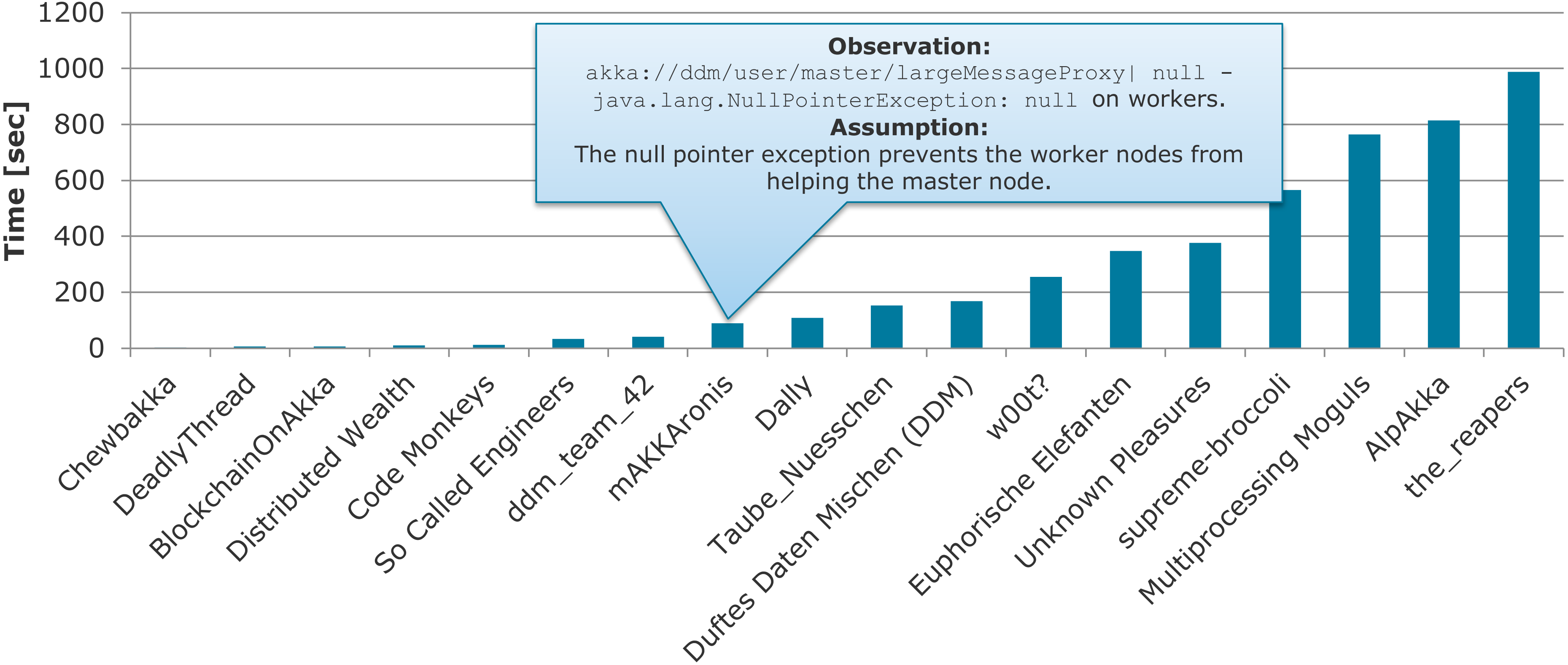



**Observation:**
Master looses all workers over time.
**Assumption:**
Master probably puts workers on too long tasks.

# How our algorithm works

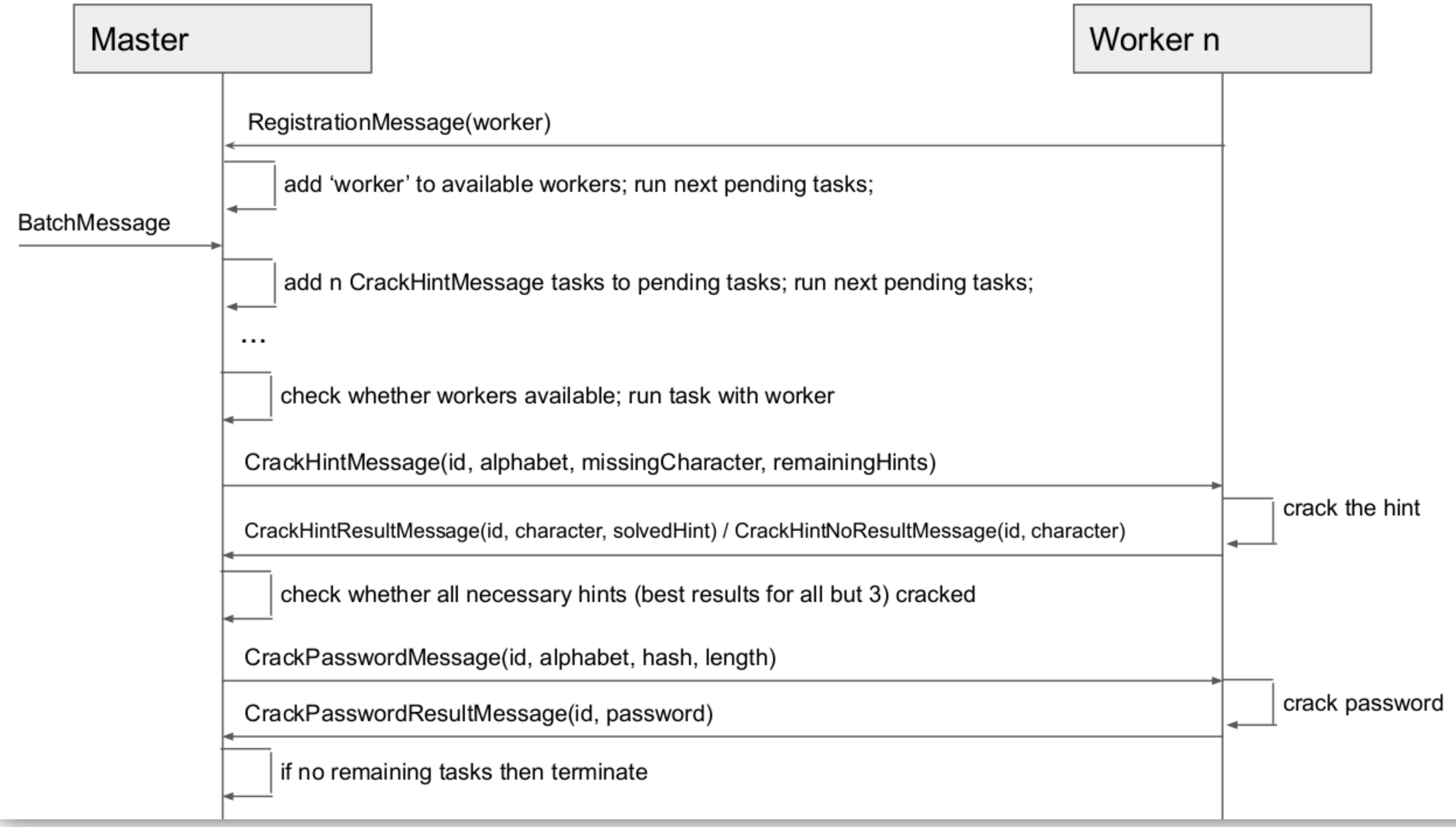
1. Master creates and registers as many workers as specified
2. Each row is sent as a task to idle workers
3. Each worker solves hints
   a. Use Heap's algorithm for generating all possible permutations where each character of the input alphabet occurs at most once
   b. Check for each permutation if its hash is equal to the hint hash -> if yes: go to next hint
4. Each worker solves password
   a. Use only the possible characters that are left after the hints to create all possible permutations with the given password length
   b. Check for equality between each hashed permutation and the hashed password → if equal: send to master
5. Master receives passwords and sends them to the Collector with their ID
6. Once all workers are finished, master terminates all workers and tells Collector to print all passwords

# Task 3 – students.csv
## 100 names; 10 length; 11 chars; 9 hints; 10 worker/node



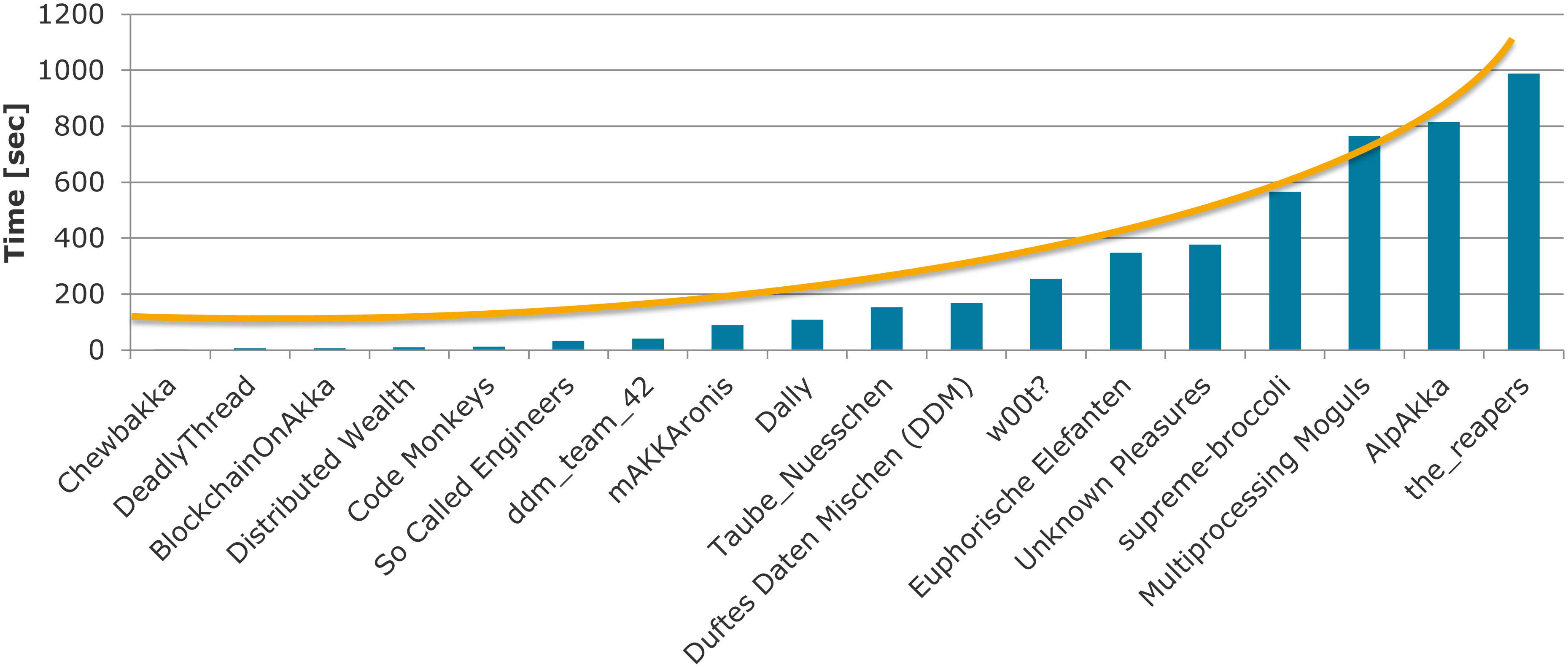
# PasswordCracking

We keep a list of pending tasks. Such a task is e.g.to crack a hint for a user. When a worker is added or another task finishes, the next pending task is executed. Tasks are added dynamically. Permutations are calculated lazily by the workers.

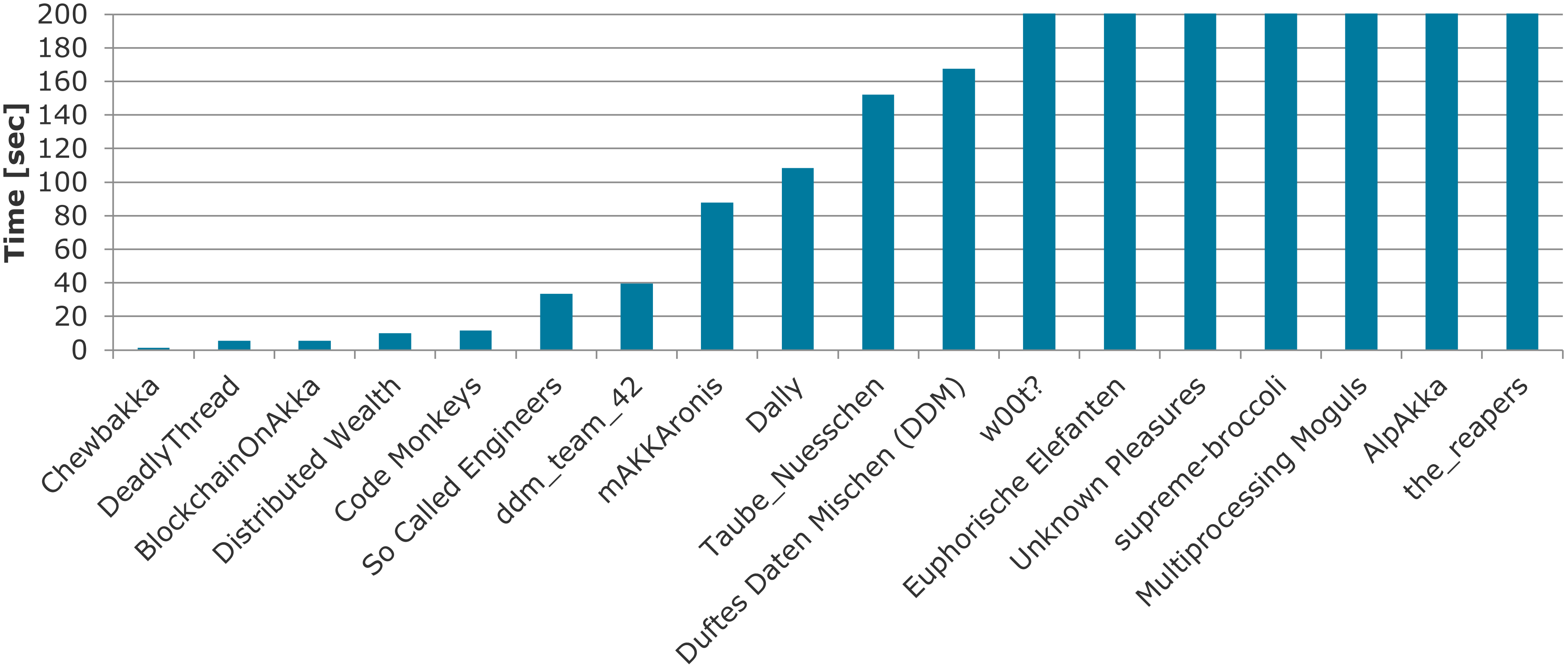


# Task 3 – students.csv
100 names; 10 length; 11 chars; 9 hints; 10 worker/node

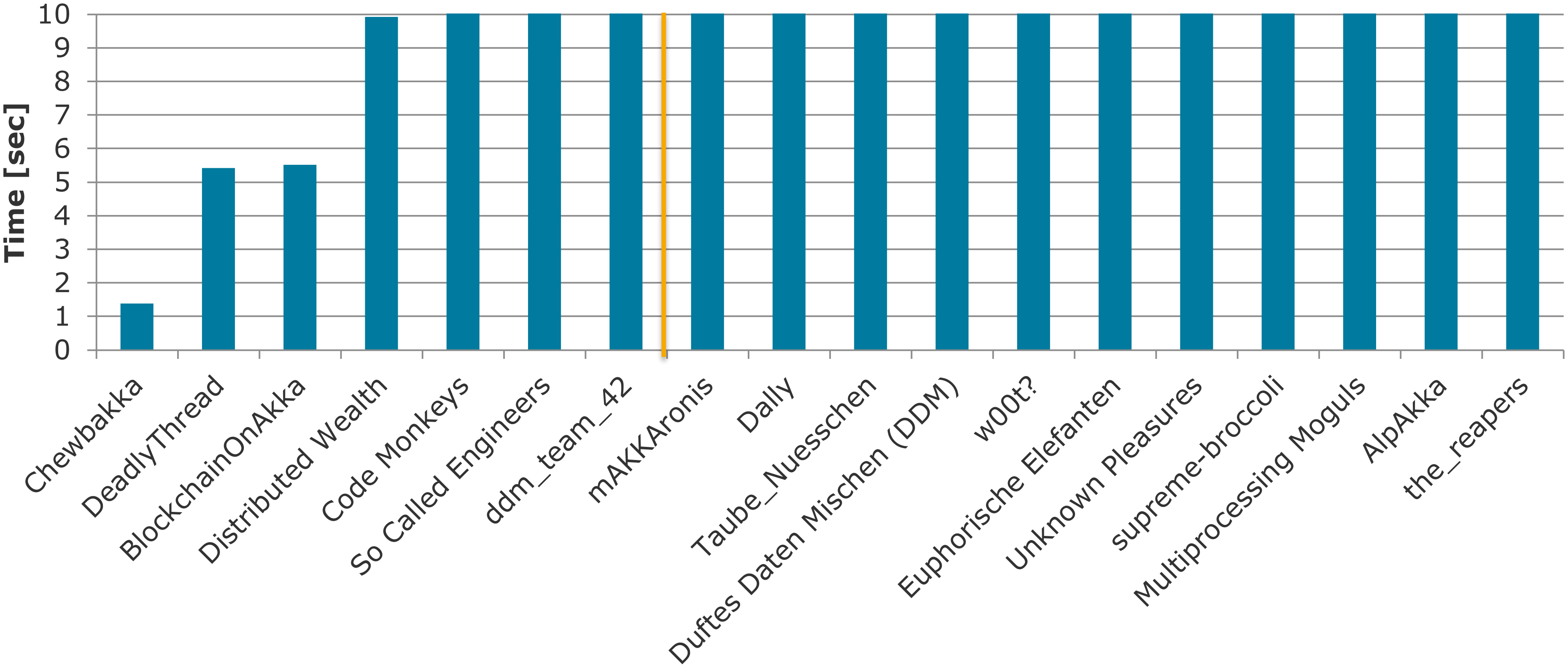




# Task 3 – students.csv
## 100 names; 10 length; 11 chars; 9 hints; 10 worker/node

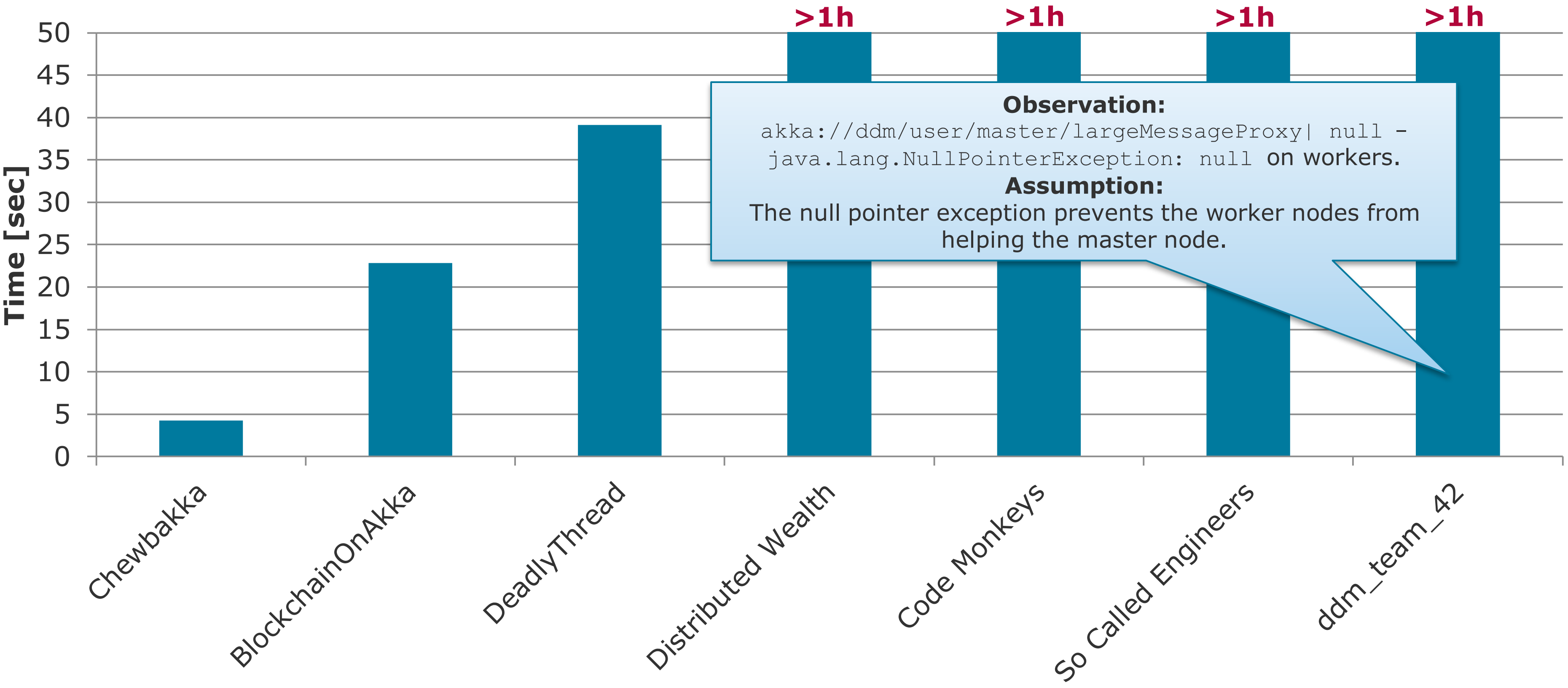


# Task 3 – students.csv
100 names; 10 length; 11 chars; 9 hints; 10 worker/node



# Task 3 – students_hard.csv

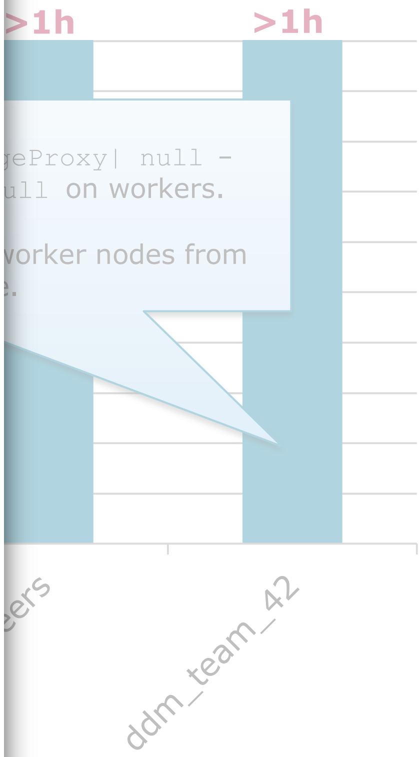
## 100 names; **12** length; **12** chars; **10** hints; 20 worker/node



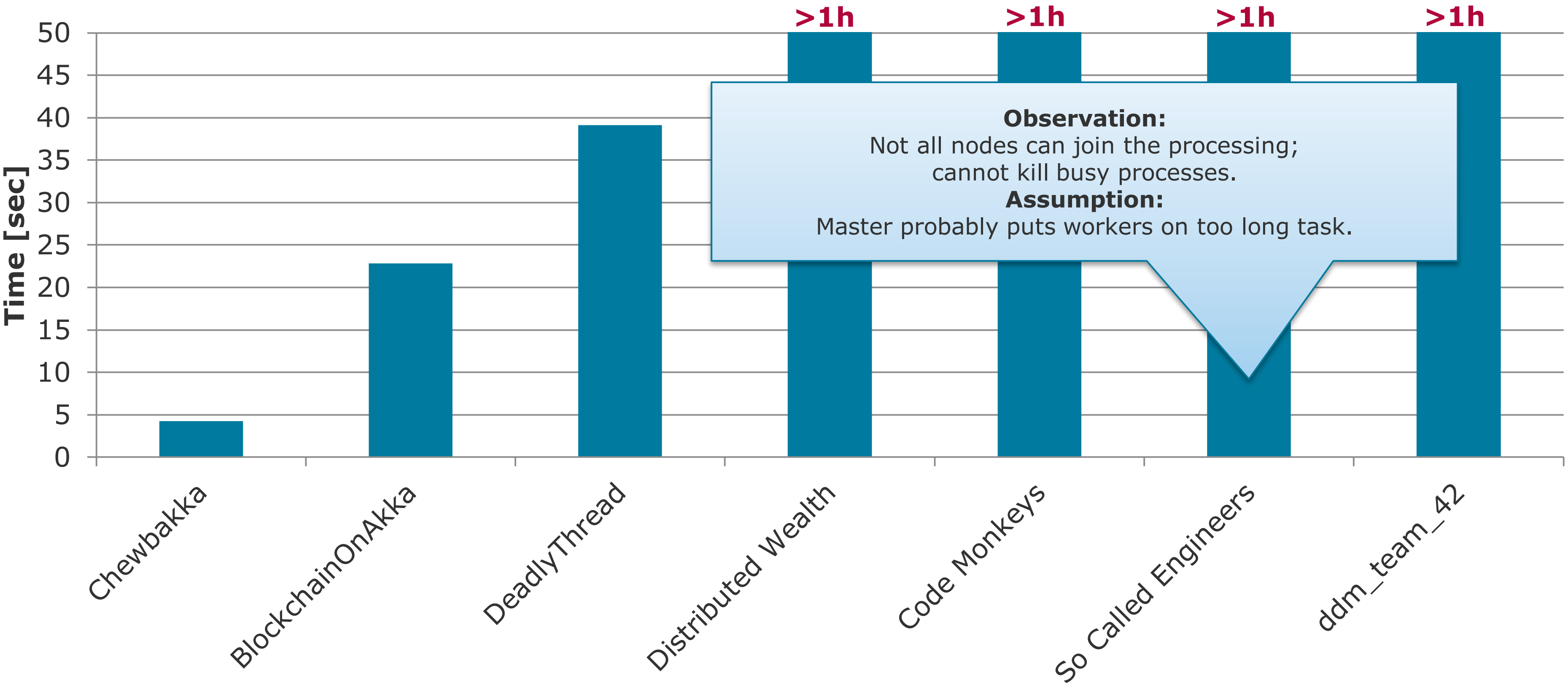
## Assignment 3 (Password cracking with Akka)

1. In handle(BatchMessage) the master receives the input batches and stores the read records into a global variable. Initially the messages for creating the permutations for the password chars (which are equal for all rows) are sent to the worker, called ***BuildPermutationsMessage***. Here one char each is left out from the set of chars, as every hint contains all password chars besides one.
2. In handle(BuildPermutationsMessage) the workers build the permutations for a given char set and hash them. Then, the workers check if the computed hashes match with the hints given in the records. All cells in which the hints match the hash and their resolved hints are sent back to the master with the ***ReceivePermutationHashMessage.***
3. In handle(ReceivePermutationHashMessage) the master reads all resolved hints which contains all cells for which the hashes where resolved. Each hint entry in the records that that was resolved gets updated with the new resolved value. Once all hints are resolved the master sends messages for each record to crack the password, called ***CrackPasswordMessage.***
4. In handle(CrackPasswordMessage) the workers crack the password of one record. For this all possible chars that are included in the password are determined by using the hints. From these chars all possible combinations are computed and hashed. The hashed combinations are compared to the password hash that is given in the record. The non-hashed password of the belonging matching hash is written to the records. The now completely resolved record is now sent to the master again via the ***ReceiveResolvedRecordMessage.***
5. In handle(ReceiveResolvedRecordMessage) the master sends the resolved record to the collector. Once all records have been resolved the terminate() method is called.

The workers are distributed via round-robin (a counter and a module function are in place).



# Task 3 – students_hard.csv
100 names; **12** length; **12** chars; **10** hints; 20 worker/node



# Cracking with “So called Engineers”

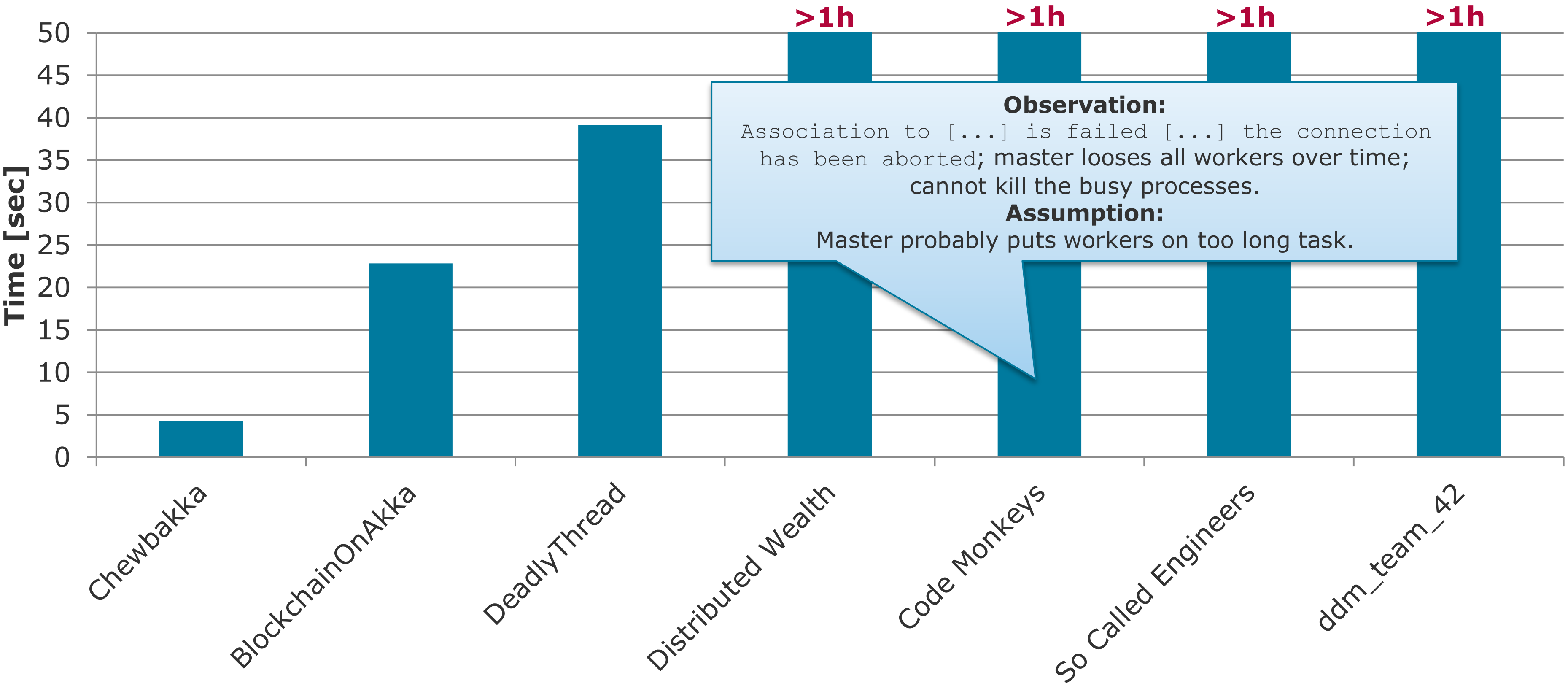
High-Level: Master hat eine TaskQueue, die alle Tasks beinhaltet

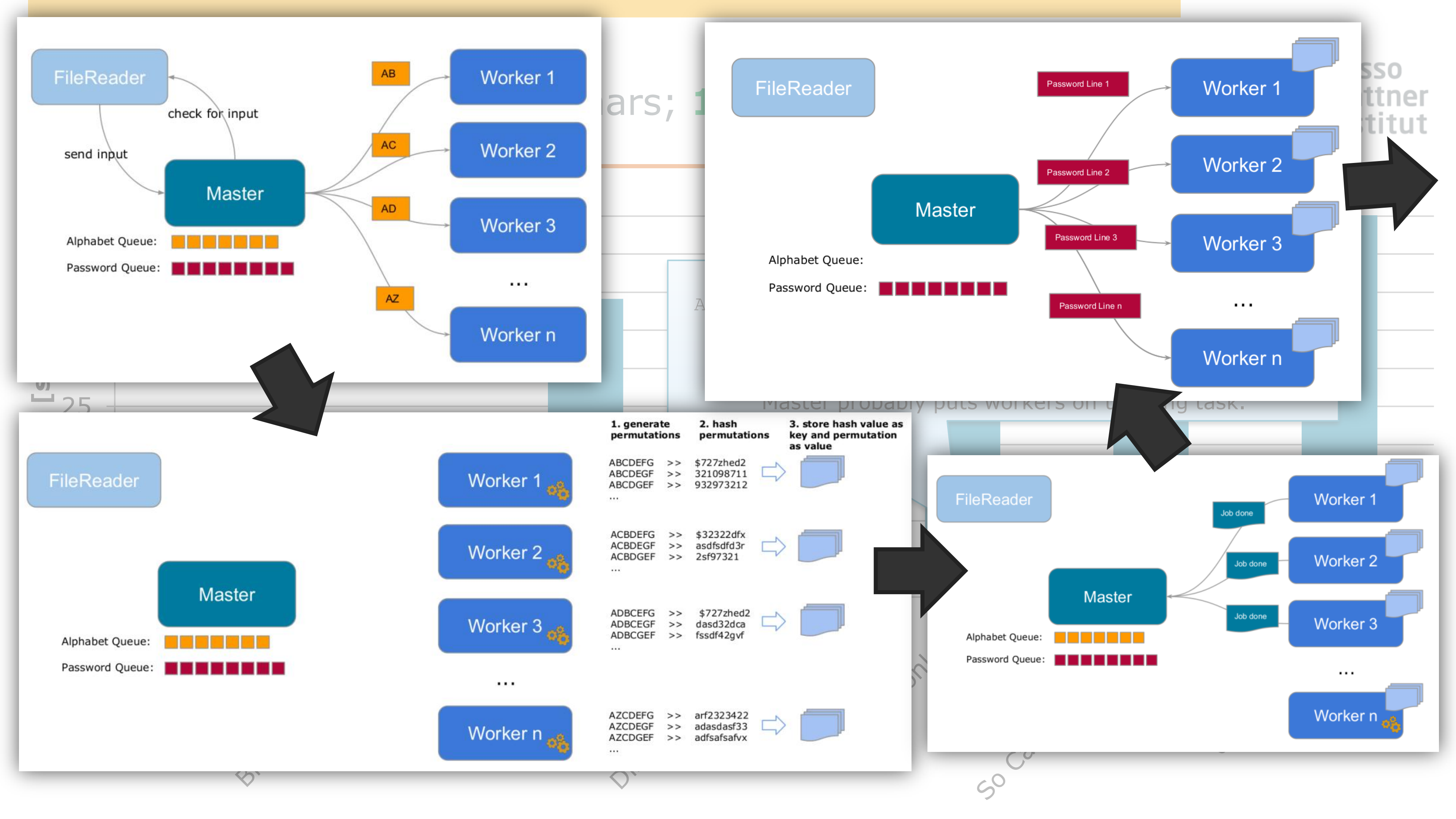
Worker kann jeden Task lösen. Schickt Ergebnisse an Master zurück, und erhält direkt neuen Task. Tasks sind “Crack diese Hints von X Usern” und “Crack diesen Hash mit Alphabet $\propto$”

Master teilt beim ersten mal HintCrackingTask auf worker auf → schickt tasks an worker

*Super krasse* Optimierung: Cracke viele Hints gleichzeitig, da alle Hints den gleichen Inputraum benutzen und wir so Hashberechnungen sparen

# Task 3 – students_hard.csv
100 names; **12** length; **12** chars; **10** hints; 20 worker/node

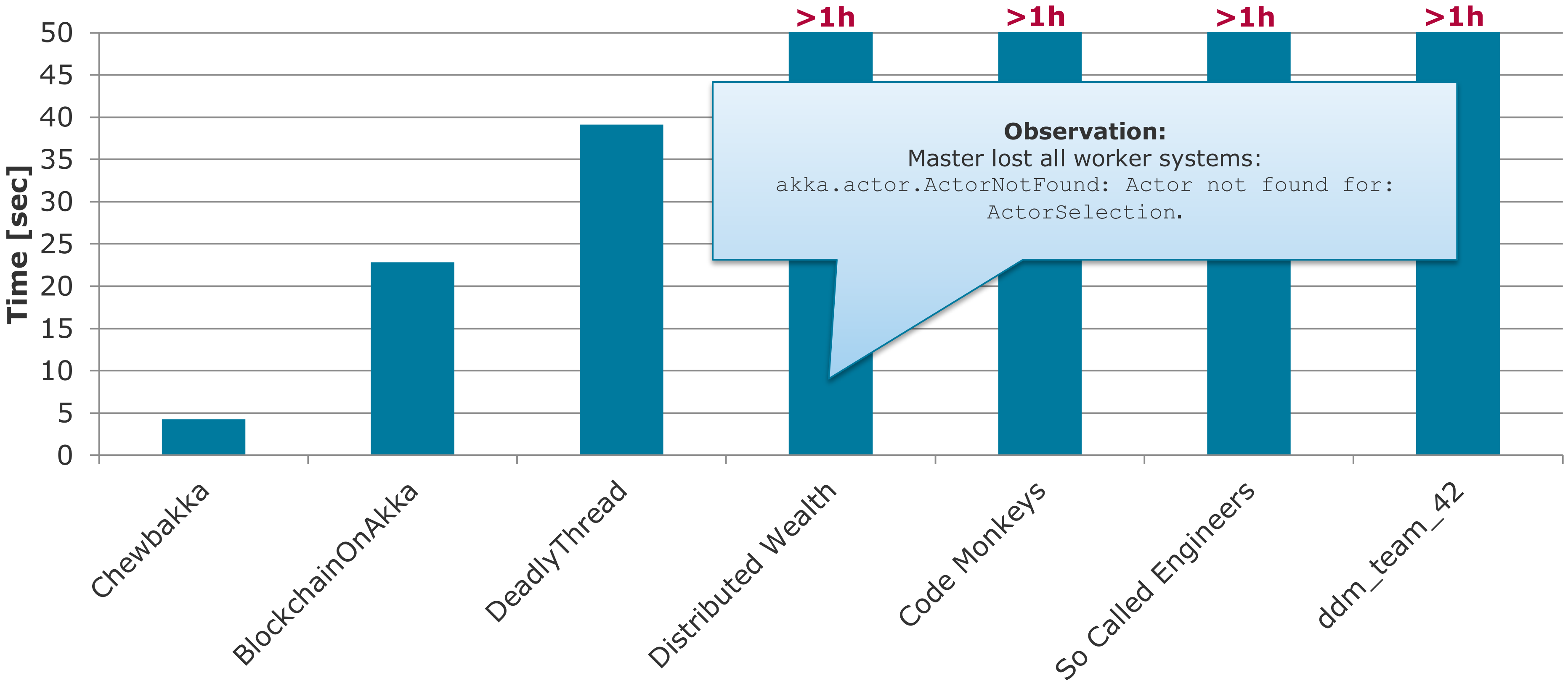




FileReader
check for input
send input
Master
Alphabet Queue:
Password Queue:
AB
AC
AD
AZ
Worker 1
Worker 2
Worker 3
...
Worker n
Password Line 1
Password Line 2
Password Line 3
Password Line n
1. generate permutations
2. hash permutations
3. store hash value as key and permutation as value
ABCDEFG >> $727zhed2
ABCDEGF >> 321098711
ABCDGEF >> 932973212
...
ACBDEFG >> $32322dfx
ACBDEGF >> asdfsdfd3r
ACBDGEF >> 2sf97321
...
ADBCEFG >> $727zhed2
ADBCEGF >> dasd32dca
ADBCGEF >> fssdf42gvf
...
AZCDEFG >> arf2323422
AZCDEGF >> adasdasf33
AZCDGEF >> adfsafsafvx
...
Job done
Job done
Job done

# Task 3 – students_hard.csv
100 names; **12** length; **12** chars; **10** hints; 20 worker/node

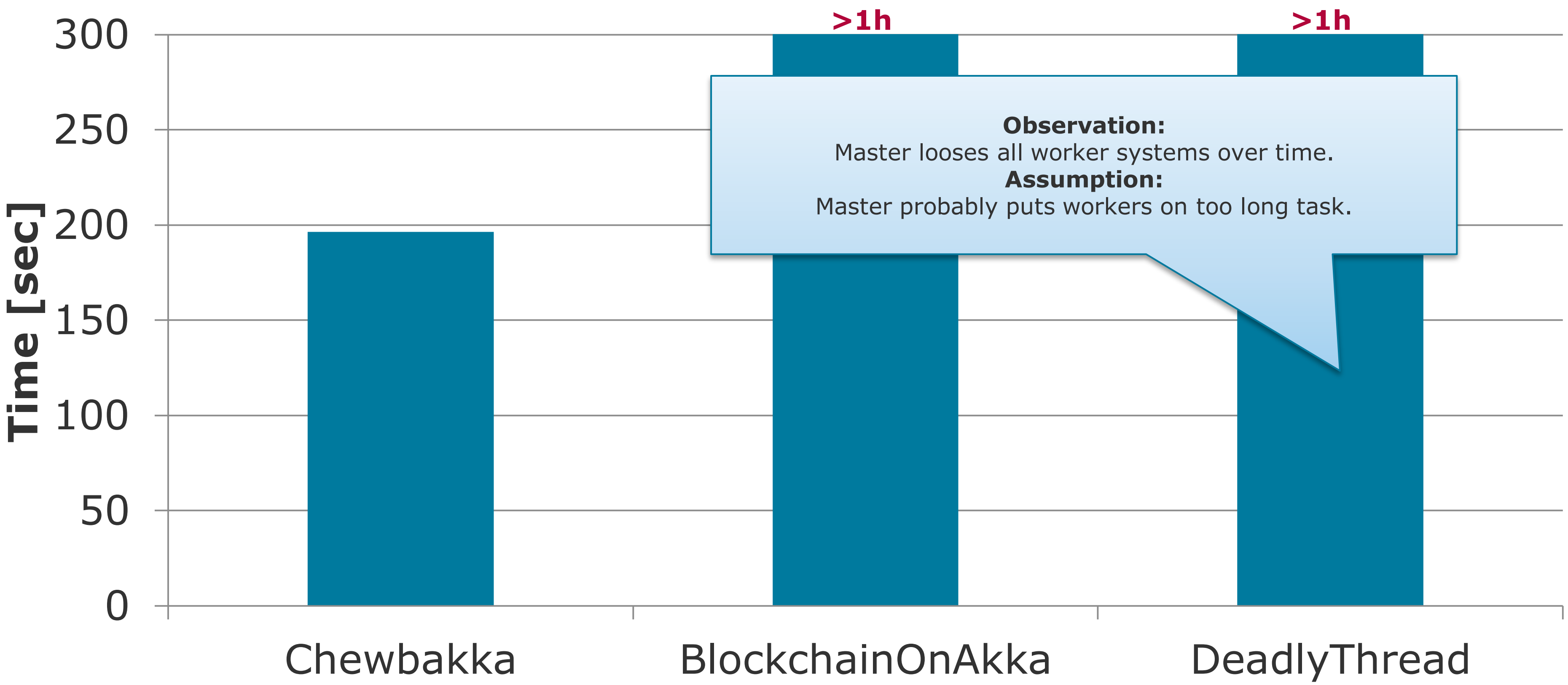




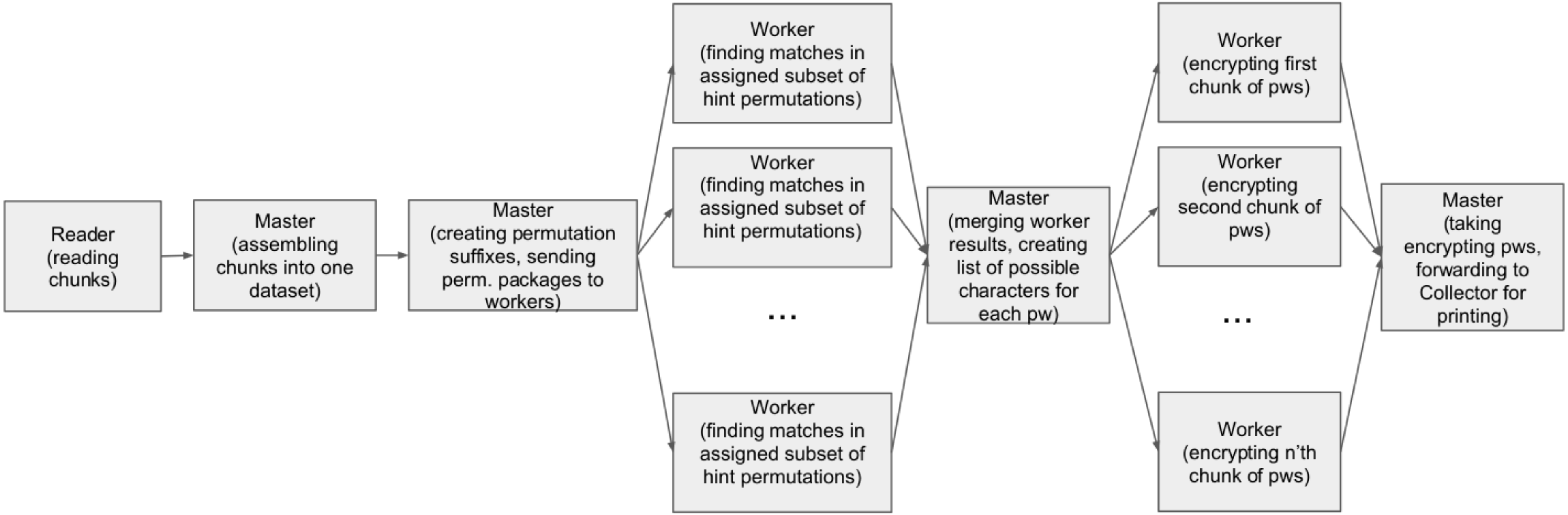
# Assignment 3

- Main idea: Each workers calculates hashes for a range of possible hints. Then, for a given hash, the worker tries all of our encrypted hints to see if one encrypted hint matches.
  - Why? Hashing is expensive(99% runtime) and we want to avoid duplicate hash calculations.
  - -> The larger the batch, the greater the performance of the algorithm
    Runtime ~ #batches
- We have one master and *n* workers
  - Master: Distributes the hint candidates to the workers. Sends all encrypted hints to every worker.
  - Worker: Calculates the hashes for all of its hint candidates and compares the encrypted hints to them.
  - Similar procedure for password decryption
- Pull Propagation: Worker sends message to master in order to request a new task.
- Tasks can either be hint or password decryption
- When all tasks for a batch are distributed, we request a new batch so that workers that finish their task can get a new task (without waiting for the others to finish)
- Once all hints for a batch are received, we add password encryption tasks for the batch. Once all passwords are encrypted, the batch is deleted from master main mem.

## Task 3 – students_extreme.csv
100 names; **8** length; **14** chars; **10** hints; 20 worker/node



# General Idea



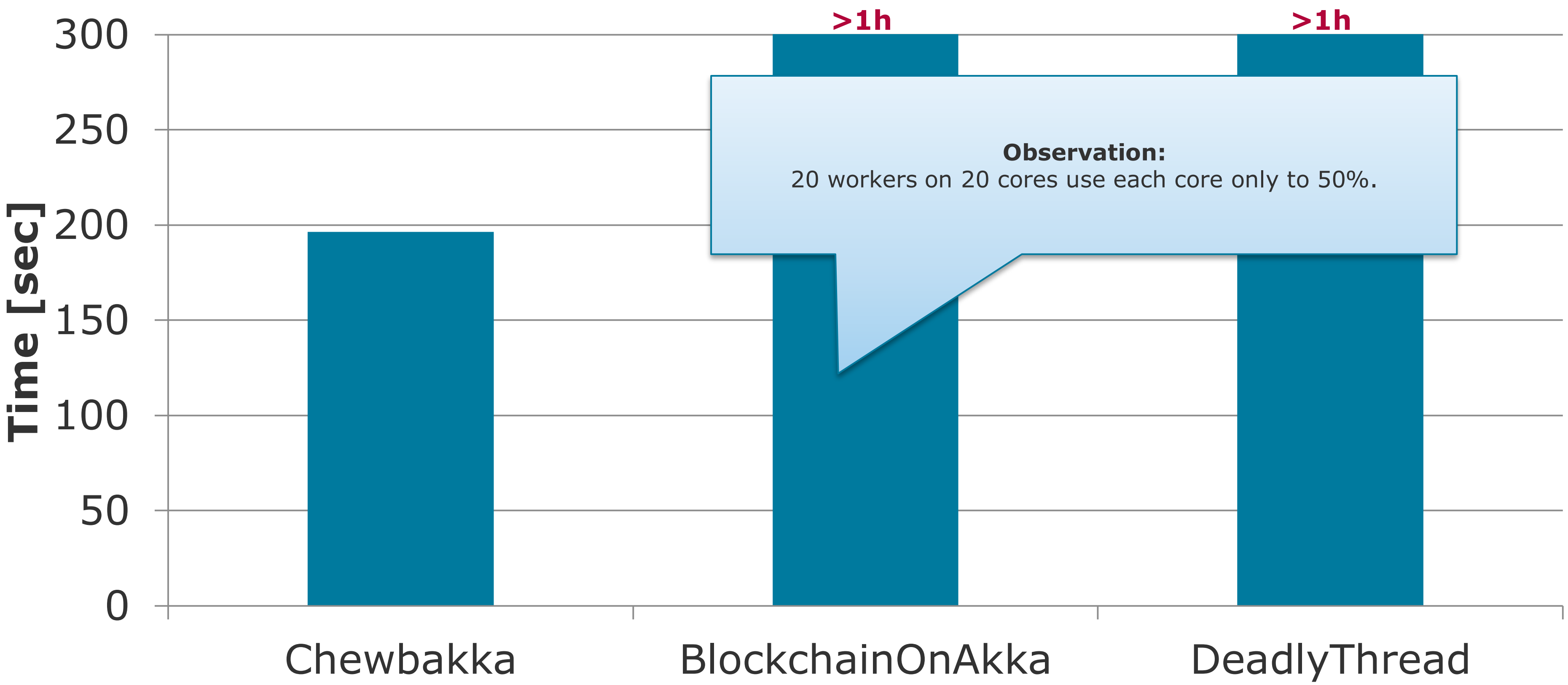
# General Idea notes

- Generate permutations suffixes on master (fast, not that many data to send)
- Send suffixes to workers
- Workers create all prefix permutations
- Workers try permutations
- Master merging results -> creating list of possible characters per password
- Workers try all combinations per password, sind result to master

# Some thoughts

- Hints are permutations >> there are not that many hints possible
  - Generating all possible hints, hashing them and comparing with available hints using a Hashmap is faster than decrypting each hint
- First decrypting all hints before starting to decrypt passwords because hints make pw problem much easier
- Parallelising hint decryption by iterating through all permutation suffixes and giving batches of permutations to workers for trying them out (hashing them and comparing to hashed hints in Hashmap)
  - Generating permutations is cheap compared to hashing them -> generating permutations at master and doing the hashing in parallel on the workers
- Assuming that there are not many duplicates in the passwords, finally constrained passwords (some of the characters were eliminated because of hints) are decrypted in parallel by assigning a small batches of pws to workers

# Task 3 – students_extreme.csv
## 100 names; **8** length; **14** chars; **10** hints; 20 worker/node



# Team BlockchainOnAkka: *Password Cracking*


```
Basic idea: 1. Crack all Hints  2. Crack all passwords
                 (shared space)      (partition per password)
Features: Clever partitioning of hint and password solution space (no
duplicate work), work stealing, local master concept to avoid unnecessary
network traffic, Apache SHA-256
```



## |Σ|! Hints |Σ|^Len Passwords

Global Master

HashSet of Hints over Network

HashSet of Hints over Network

ActorSystem

Local Master / Worker

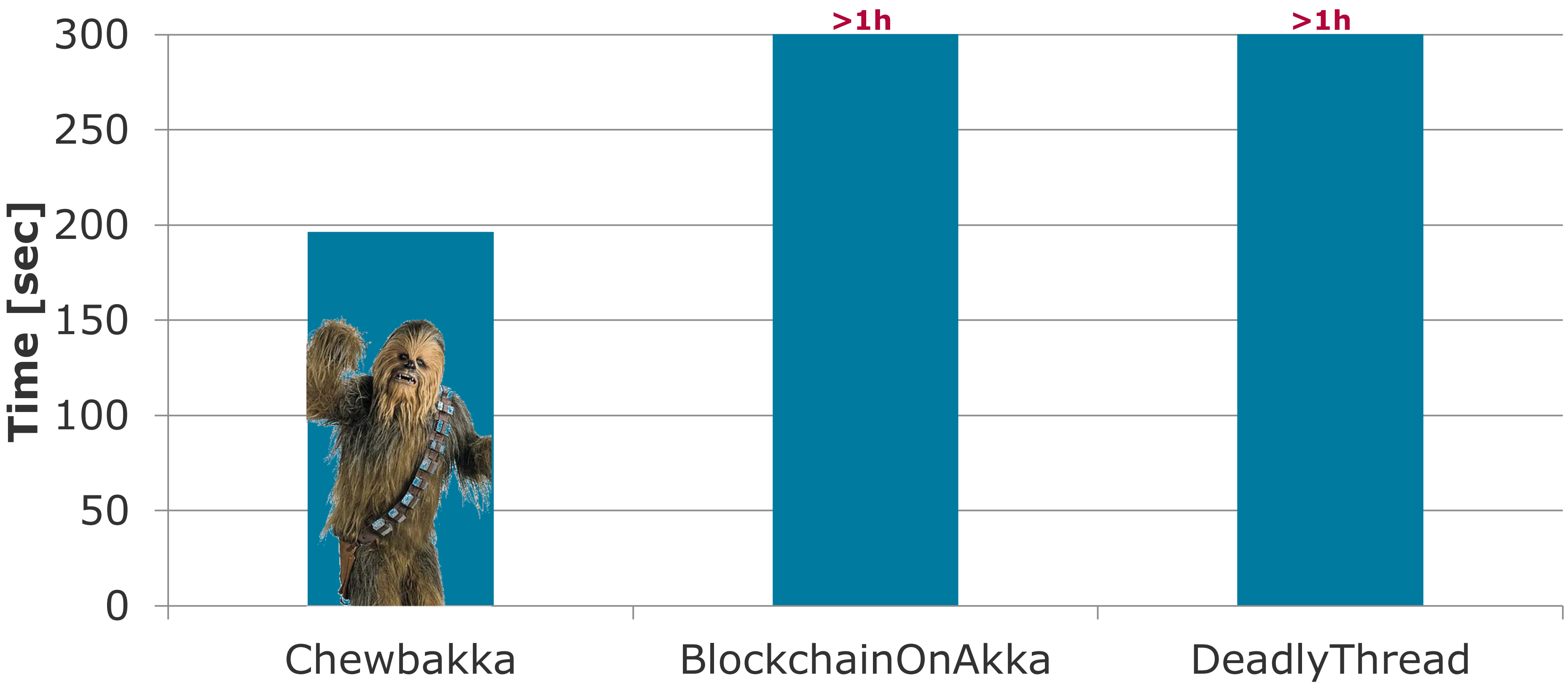
HashSet of Hints (shared memory)

Many Workers

Local Master / Worker




## Task 3 – students_extreme.csv
100 names; **8** length; **14** chars; **10** hints; 20 worker/node



# 2. Password Cracking (1/5)

- Hints are cool, but does it always make sense to crack them?
  - Task:
    - The more hints we have, the easier it is to find the password.
  - Each hint allows one more character to exclude:
    $$\#unqiueCharsInPassword = passwordLength - \#hints$$
    Example: password length of 11, 9 hints → password consists of 2 different characters
  - Difficulty (worst-case, max. number of hashes) of cracking a hint?
    $$D_{Hint} = (\#charsInAlphabet - \#crackedHints) \cdot (\#charsinAlphabet - 1)!$$

For each cracked hint, we can exclude the already known excluded characters.

Each hint is a permutation of one less character than the alphabet.

# 2. Password Cracking (2/5)

- Difficulty of cracking a password?

$$leftoverChars = (\#charsInAlphabet - \#crackedHints)$$

$$D_{Password} = \frac{(leftoverChars)!}{\#uniquePasswordChars! \cdot (\#hints - \#crackedHints)!} \cdot \#uniquePasswordChars^{passwordLength}$$

# 2. Password Cracking (3/5)

- Difficulty of cracking a password?

$$leftoverChars = (\#charsInAlphabet - \#crackedHints)$$

$$D_{Password} = \frac{(leftoverChars)!}{\#uniquePasswordChars! \cdot (\#hints - \#crackedHints)!} \cdot \#uniquePasswordChars^{passwordLength}$$

**DDM Exercise: Akka-Handson**

Team: ChewbAKKA
Timofei Kornev
Felix Gohla
Chart **5**

# 2. Password Cracking (4/5)

- Difficulty of cracking a password?

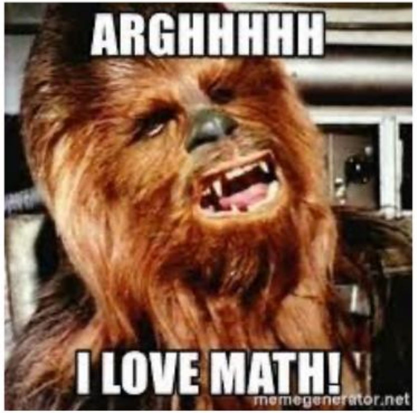
$$leftoverChars = (\#charsInAlphabet - \#crackedHints)$$

$$D_{Password} = \frac{(leftoverChars)!}{\#uniquePasswordChars! \cdot (\#hints - \#crackedHints)!} \cdot \#uniquePasswordChars^{passwordLength}$$

- Simple example:
  - 11 chars in alphabet, password length of 10,
    9 hints (0 cracked) → 2 unique characters
  - $\frac{(11-0)!}{2! \cdot (9-0)!} \cdot 2^{10} = 56320$ combinations **vs.**
    $(11-0) \cdot (11-1)! = 39916800$ for cracking the first hint

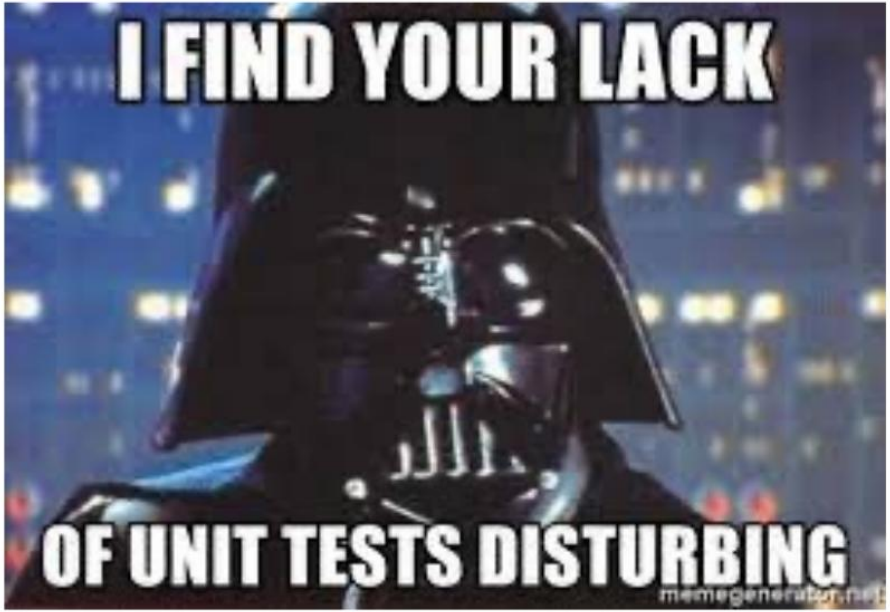
# 2. Password Cracking (5/5)

- Further improvements:
  - When there are less passwords than workers, assign already cracking passwords to the free workers.
  - They probe the combinations in a random order to not just waste energy.
- Cracks the given small dataset in ~2 seconds.



# 3. Tests

- Eeeeeeeehhhhh… wellllllll… :D



## Task 3 – students_extreme.csv
100 names; **8** length; **14** chars; **10** hints; 20 worker/node