Using Knowledge Units of Programming Languages to Recommend Reviewers for Pull Requests: An Empirical Study

Md Ahasanuzzaman

Gustavo A. Oliva

Ahmed E. Hassan

Queen's University

Software Analysis & Intelligence Lab

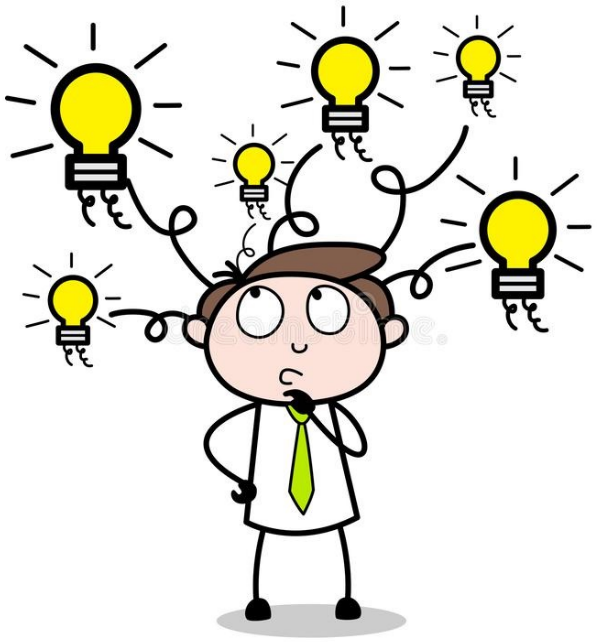
Finding the right reviewer for reviewing a piece of code is a difficult task

Finding the right reviewer for reviewing a piece of code is a difficult task

The quality of a code review inherently depends on the selection of the reviewer

Finding the right reviewer for reviewing a piece of code is a difficult task

The quality of a code review inherently depends on the selection of the reviewer



Finding the right reviewer for a set of code changes is always a **nontrivial task**, especially for a **large-scal**e, **distributed software development**

Mapping different expertise to individual developers is a key requirement for effective code review

Mapping different expertise to individual developers is a key requirement for effective code review

Recent research studies have attempted to develop approaches to detect experts in specific topics [1]

[1] Identifying experts in software libraries and frameworks among GitHub users, MSR, 2016

We focus on a key facet of expertise, which is programming language (PL) expertise

We focus on a key facet of expertise, which is programming language (PL) expertise

Our rationale is that a piece of code involving **concurrency** is suitable to be reviewed by someone who has **demonstrated experience** in dealing with **such type of code**

To capture the PL expertise of developers, we introduce the notion of Knowledge Units (KUs) of PL

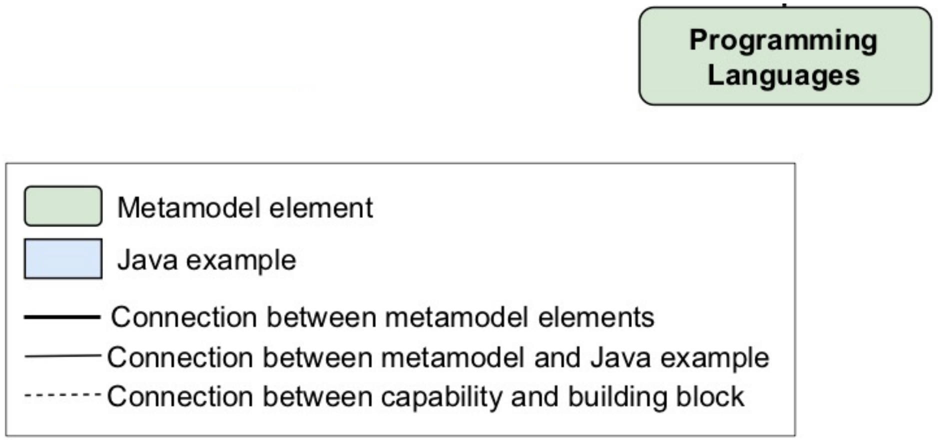
To capture the PL expertise of developers, we introduce the notion of Knowledge Units (KUs) of PL

Knowledge Unit (KU):

A Knowledge Unit (KU) is a cohesive set of **key capabilities** that are offered by one or more building blocks of a given programming language

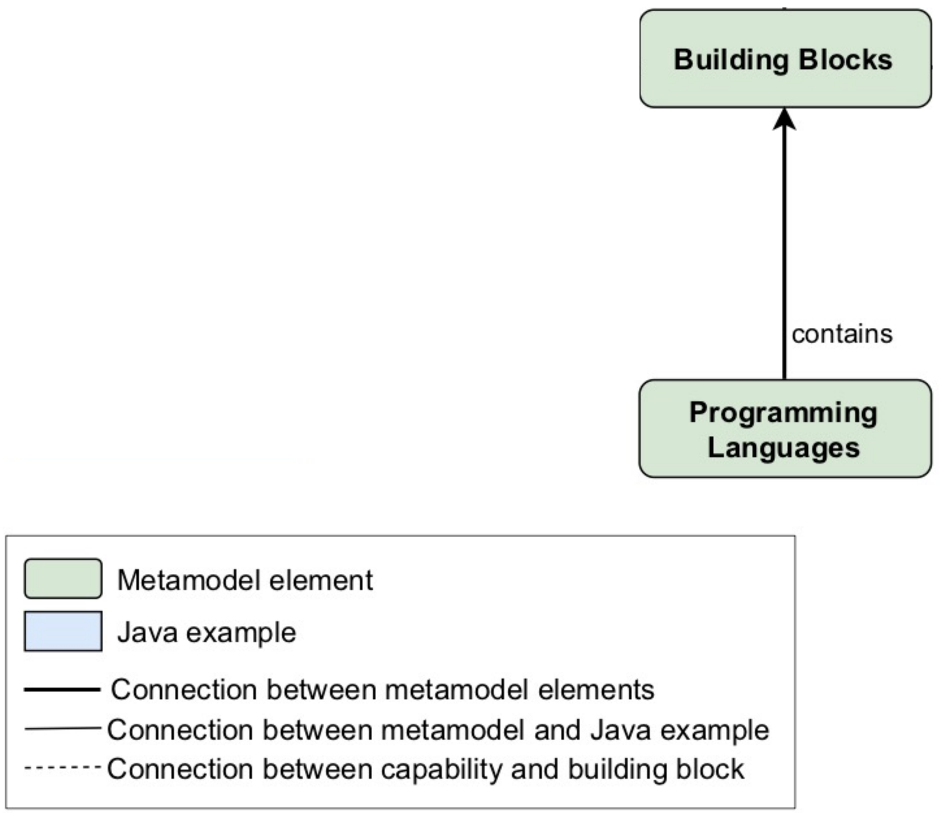
Knowledge Unit (KU):

A Knowledge Unit (KU) is a cohesive set of **key capabilities** that are offered by one or more building blocks of a given



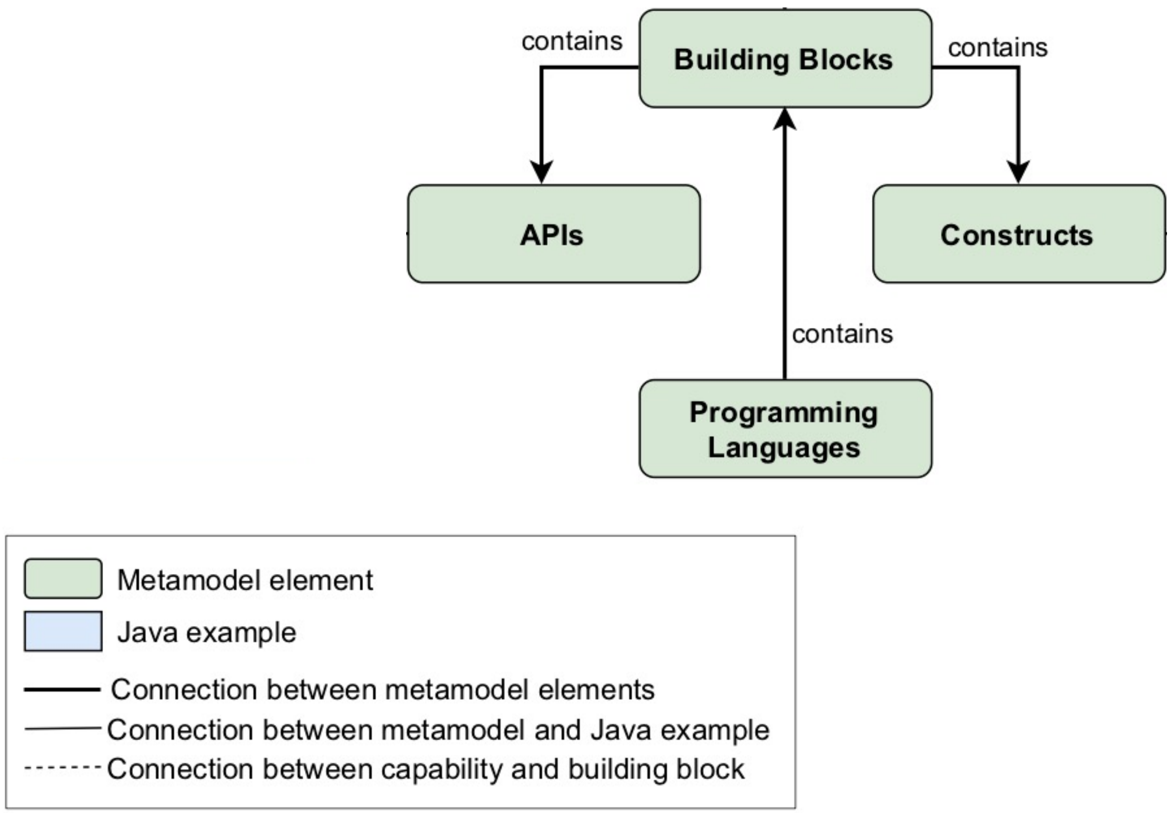
Knowledge Unit (KU):

A Knowledge Unit (KU) is a cohesive set of **key capabilities** that are offered by one or more building blocks of a given



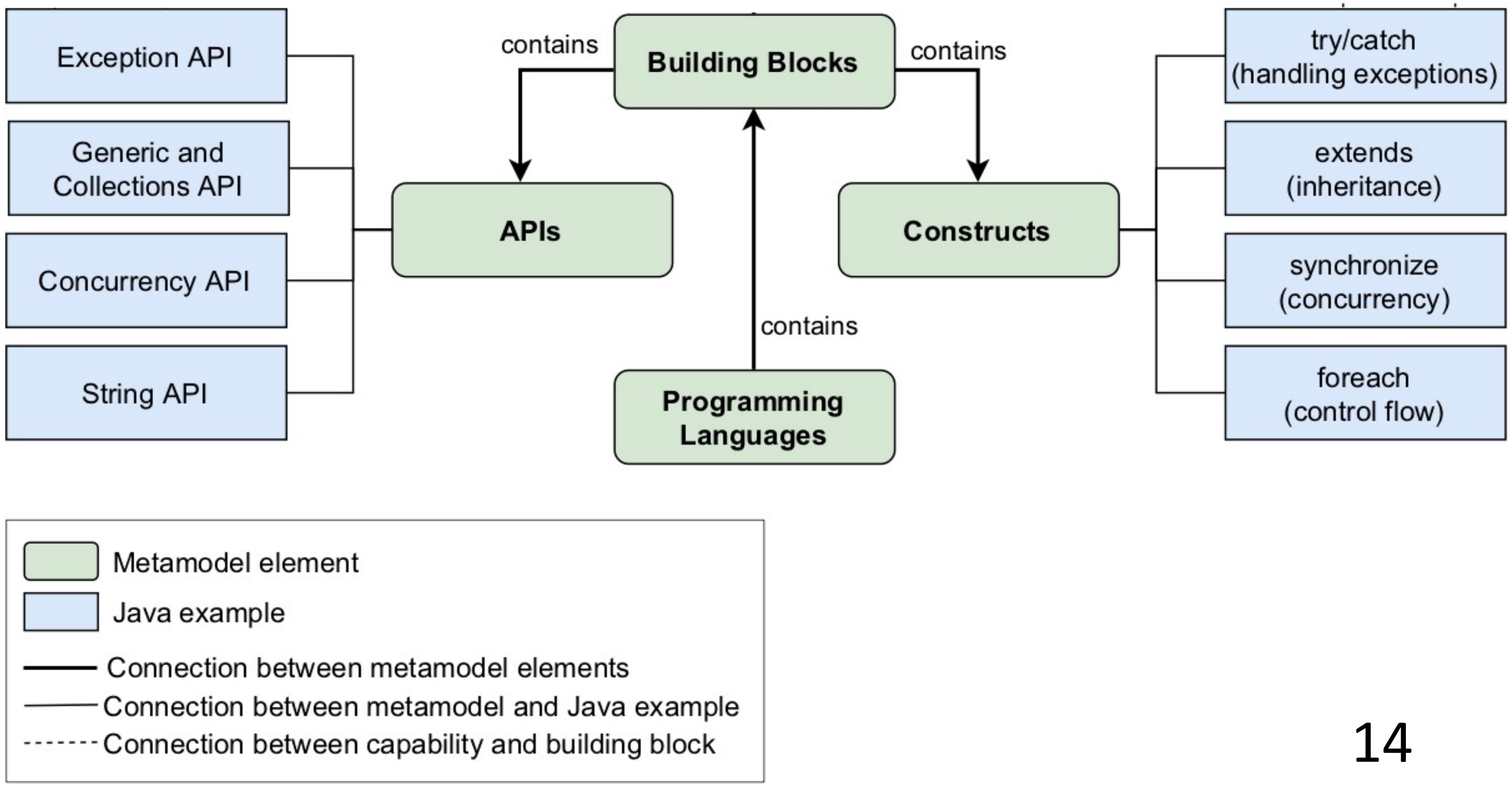
Knowledge Unit (KU):

A Knowledge Unit (KU) is a cohesive set of **key capabilities** that are offered by one or more building blocks of a given



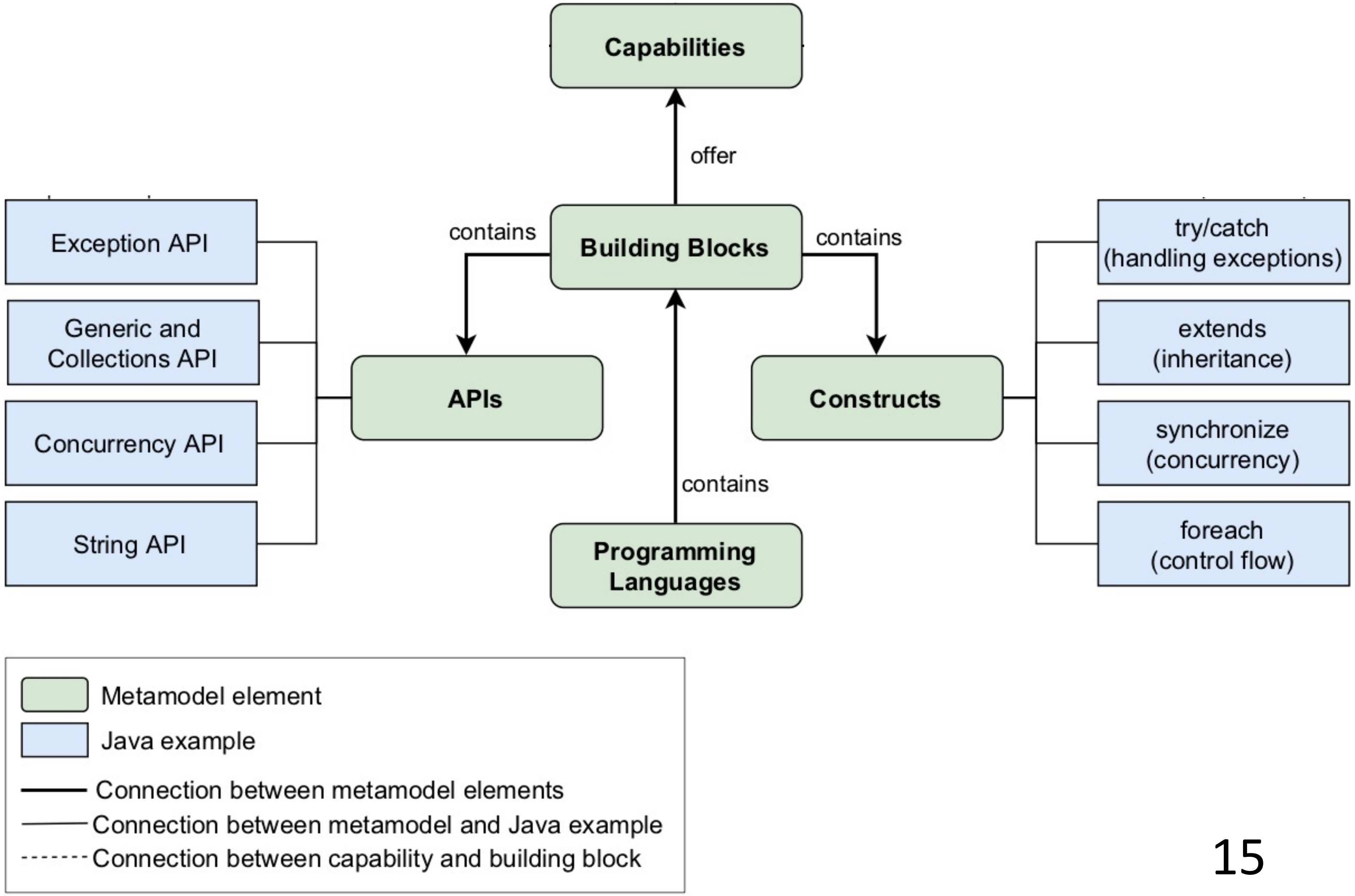
Knowledge Unit (KU):

A Knowledge Unit (KU) is a cohesive set of **key capabilities** that are offered by one or more building blocks of a given



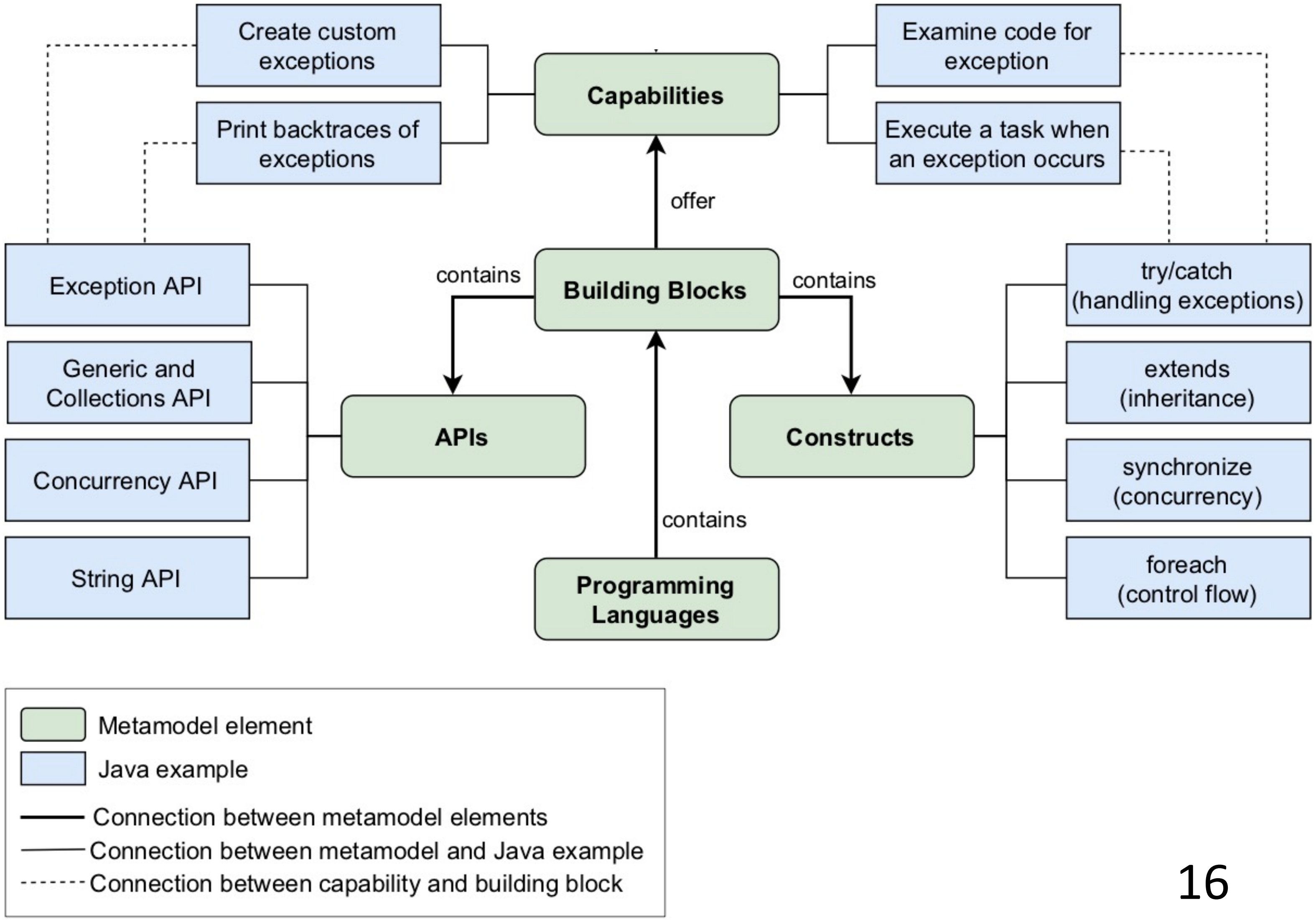
Knowledge Unit (KU):

A Knowledge Unit (KU) is a cohesive set of **key capabilities** that are offered by one or more building blocks of a given



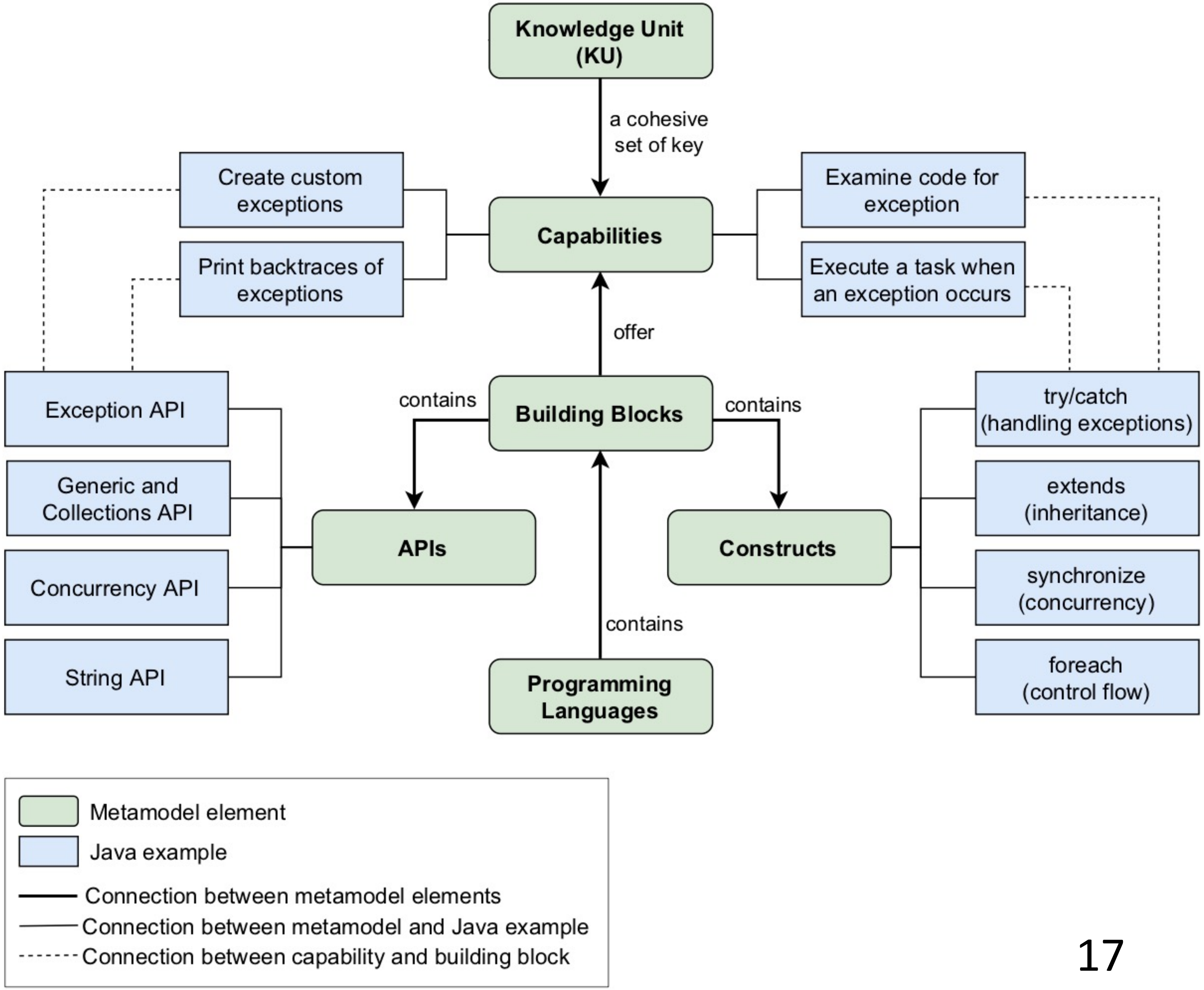
Knowledge Unit (KU):

A Knowledge Unit (KU) is a cohesive set of **key capabilities** that are offered by one or more building blocks of a given



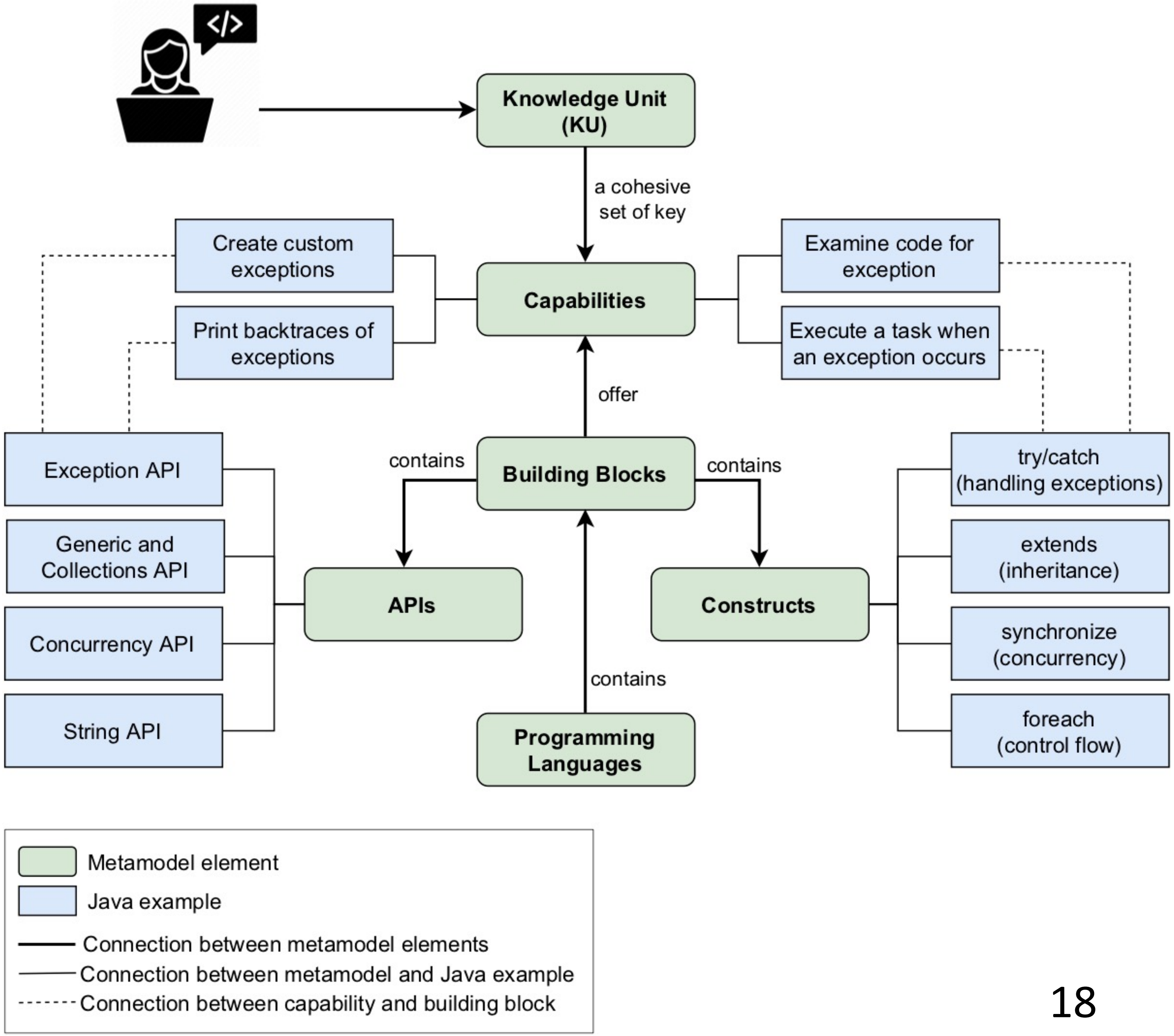
Knowledge Unit (KU):

A Knowledge Unit (KU) is a cohesive set of **key capabilities** that are offered by one or more building blocks of a given



Knowledge Unit (KU):

A Knowledge Unit (KU) is a cohesive set of **key capabilities** that are offered by one or more building blocks of a given



We operationalize our KUs via certification exams for the Java programming language

We operationalize our KUs via certification exams for the Java programming language

Oracle Java SE and Java EE certification exams for the Java

We operationalize our KUs via certification exams for the Java programming language

Oracle Java SE and Java EE certification exams for the Java

Certification exams of a programming language aim to determine the **skills and knowledge** of a developer in using the key capabilities offered by the building blocks of that language

We operationalize our KUs via certification exams for the Java programming language

Oracle Java SE and Java EE certification exams for the Java

Certification exams of a programming language aim to determine the **skills and knowledge** of a developer in using the key capabilities offered by the building blocks of that language

Thus, certification exams capture the KUs of a programming language

We map the topics and subtopics of the Java Certification Exam into KUs

Oracle University | Training | Certification | Solutions

Java Concurrency

- ✓ Create worker threads using Runnable, Callable and use an ExecutorService to concurrently execute tasks
- ✓ Identify potential threading problems among deadlock, starvation, livelock, and race conditions
- ✓ Use synchronized keyword and java.util.concurrent.atomic package to control the order of thread execution

Building Database Applications with JDBC

- ✓ Describe the interfaces that make up the core of the JDBC API including the Driver, Connection, Statement, and ResultSet interfaces and their relationship to provider implementations
- ✓ Identify the components required to connect to a database using the DriverManager class including the JDBC URL

We map the topics and subtopics of the Java Certification Exam into KUs

Oracle University | Training | Certification | Solutions

Java Concurrency

Topic

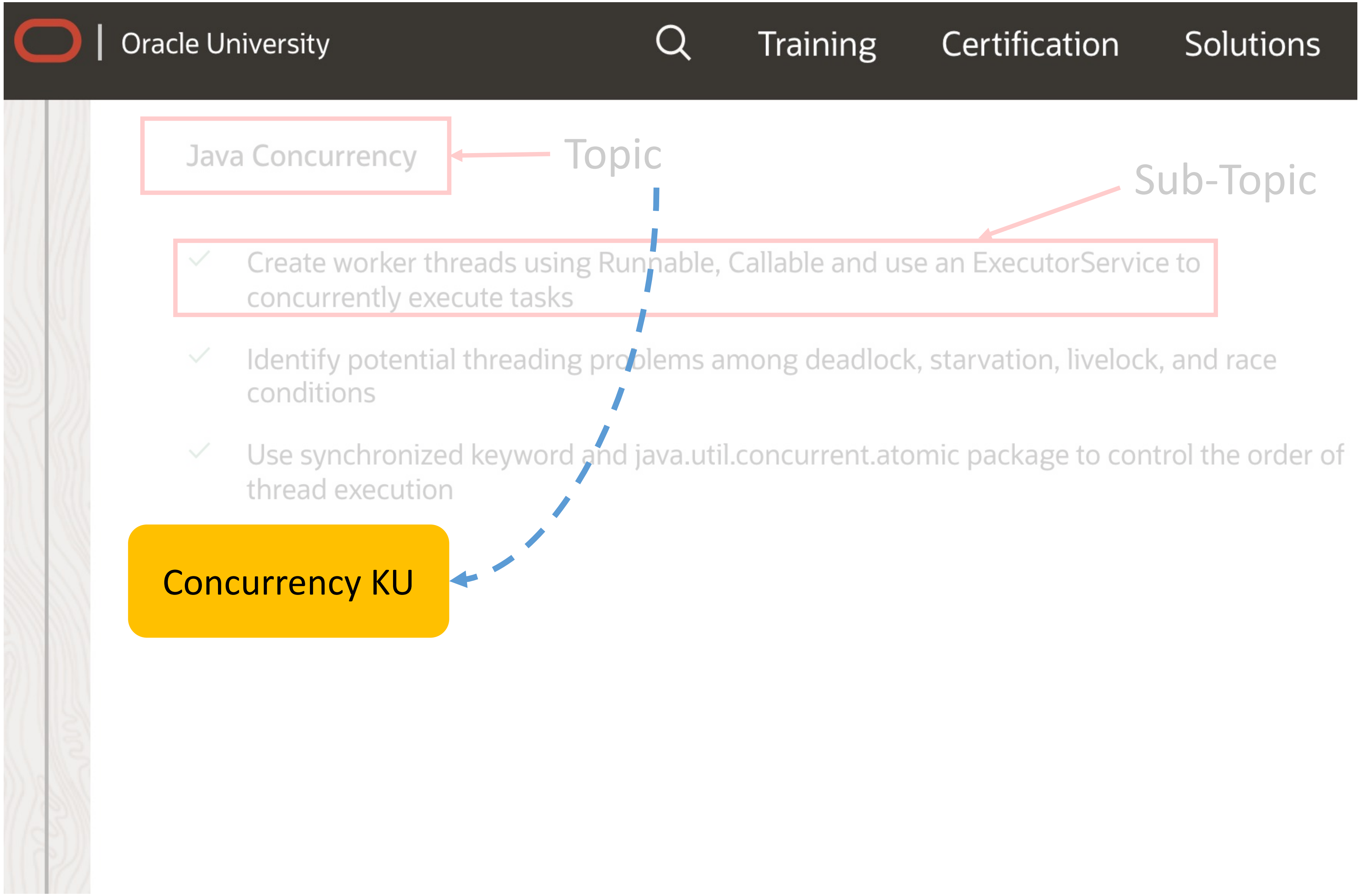
Sub-Topic

- Create worker threads using Runnable, Callable and use an ExecutorService to concurrently execute tasks
- Identify potential threading problems among deadlock, starvation, livelock, and race conditions
- Use synchronized keyword and java.util.concurrent.atomic package to control the order of thread execution

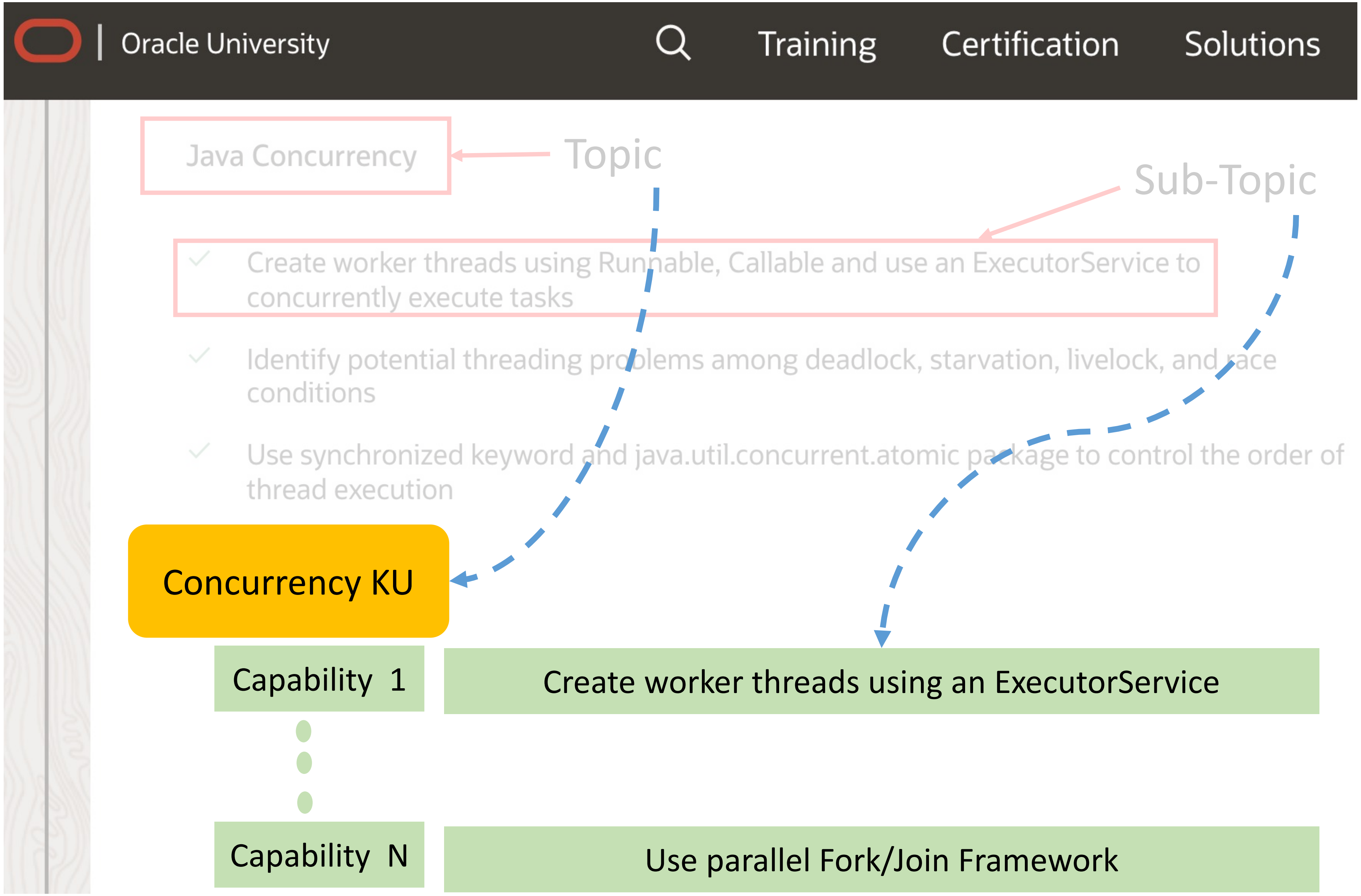
Building Database Applications with JDBC

- Describe the interfaces that make up the core of the JDBC API including the Driver, Connection, Statement, and ResultSet interfaces and their relationship to provider implementations
- Identify the components required to connect to a database using the DriverManager class including the JDBC URL

We map the topics and subtopics of the Java Certification Exam into KUs



We map the topics and subtopics of the Java Certification Exam into KUs



We identify 28 KUs of the Java programming language

We identify 28 KUs of the Java programming language

(1) Data Type KU

(2) Operator and Decision KU

(3) Array KU

(4) Loop KU

(5) Method and Encapsulation KU

(6) Inheritance KU

(7) Advanced Class Design KU

(8) Generics and Collection KU

(9) Functional Interface KU

(10) Stream API KU

(11) Exception KU

(12) Date time API KU

(13) IO KU

(14) NIO KU

(15) Concurrency KU

(16) Database KU

(17) String Processing KU

(18) Localization KU

JAVA SE KUs

We identify 28 KUs of the Java programming language

(1) Data Type KU

(2) Operator and Decision KU

(3) Array KU

(4) Loop KU

(5) Method and Encapsulation KU

(6) Inheritance KU

(7) Advanced Class Design KU

(8) Generics and Collection KU

(9) Functional Interface KU

(10) Stream API KU

(11) Exception KU

(12) Date time API KU

(13) IO KU

(14) NIO KU

(15) Concurrency KU

(16) Database KU

(17) String Processing KU

(18) Localization KU

JAVA SE KUs

(19) Java Persistence KU

(20) Enterprise Java Bean KU

(21) Java Message Service API KU

(22) SOAP Web Service KU

(23) Servlet KU

(24) Java REST API KU

(25) Websocket KU

(26) Java Server Faces KU

(27) Contexts and Dependency injection (CDI) KU

(28) Batch Processing KU

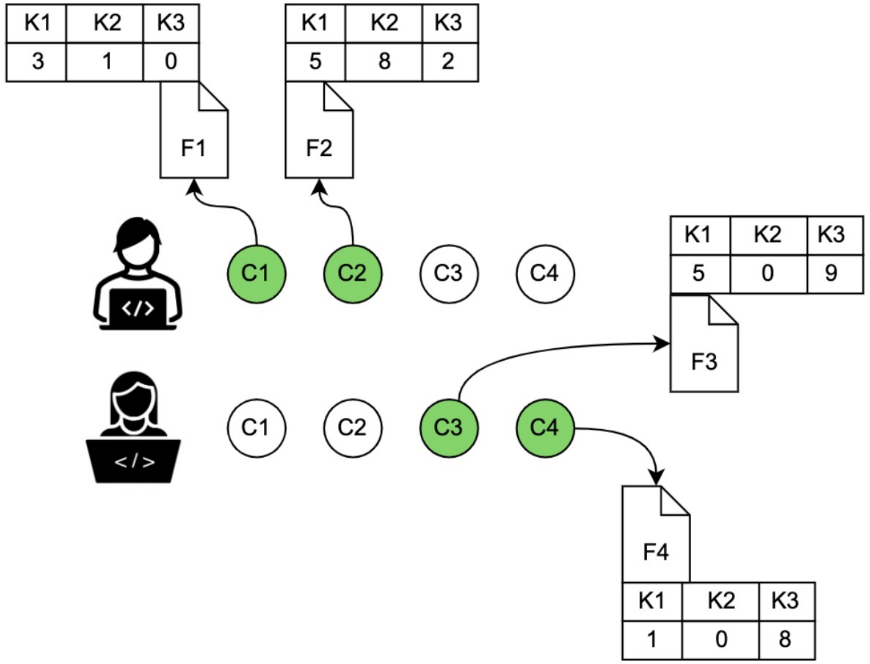
JAVA EE KUs

Our objective

How we can leverage KUs to build expertise-profile for developers and construct a **recommender system (KUREC)** for GitHub pull requests (PRs)

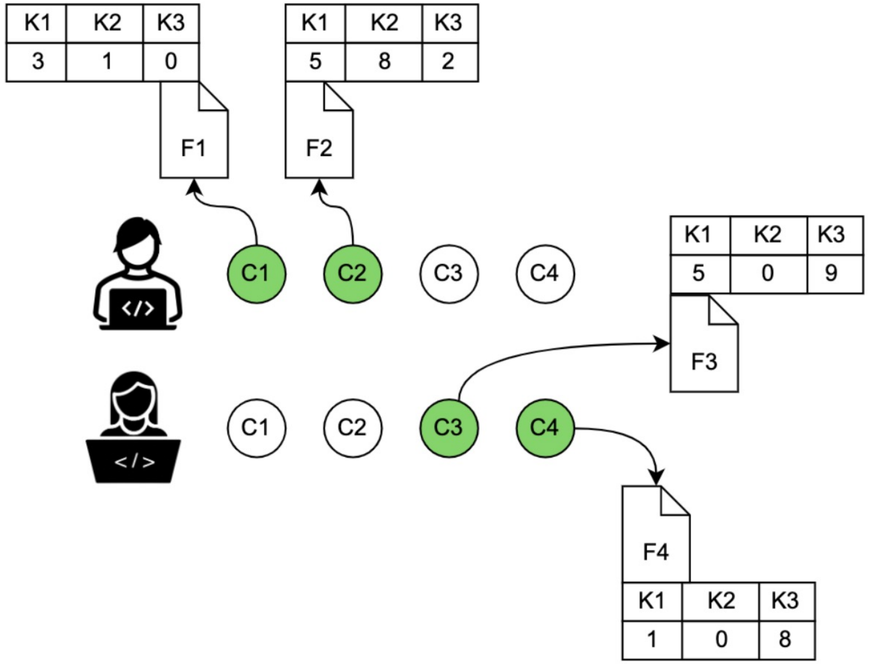
We represent developers’ expertise with KUs that are associated with changed files in commits

We represent developers' expertise with KUs that are associated with changed files in commits

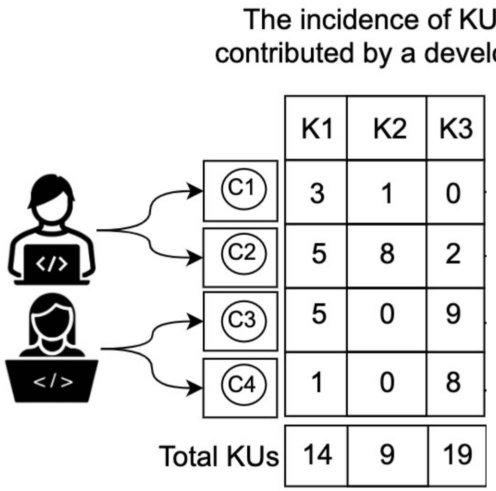


(a) Developers' commit activity and the changed files

We represent developers' expertise with KUs that are associated with changed files in commits

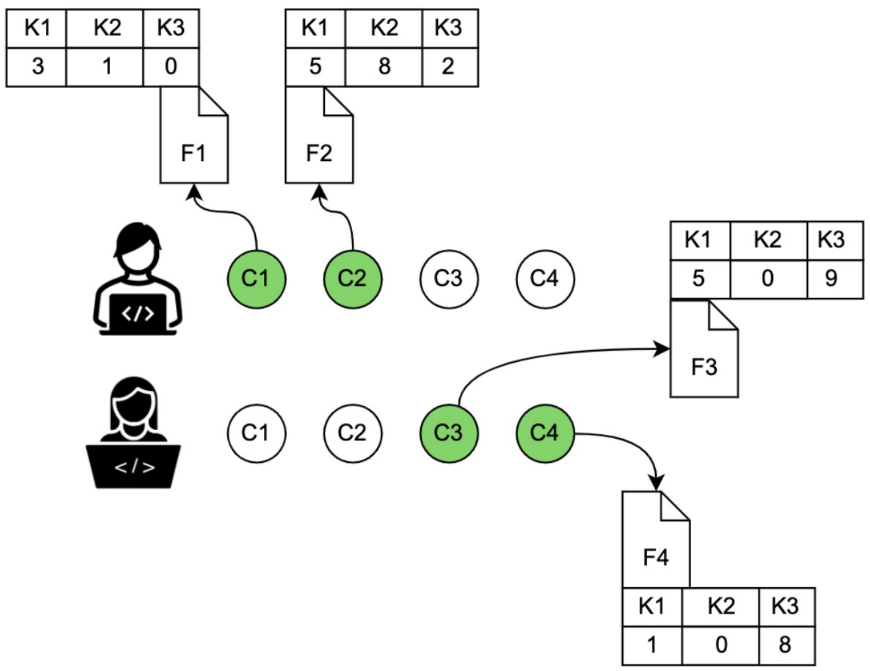


(a) Developers' commit activity and the changed files

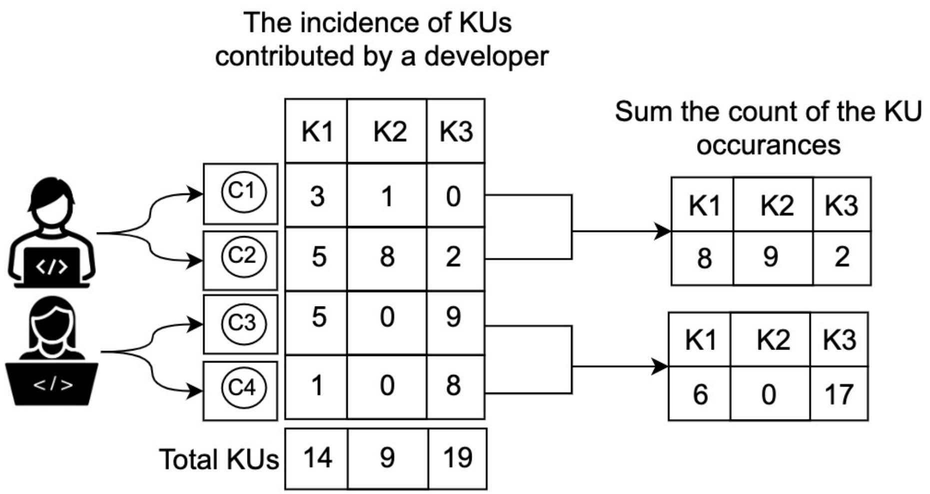


(b) Representation of developer's expertise with KUs

We represent developers' expertise with KUs that are associated with changed files in commits

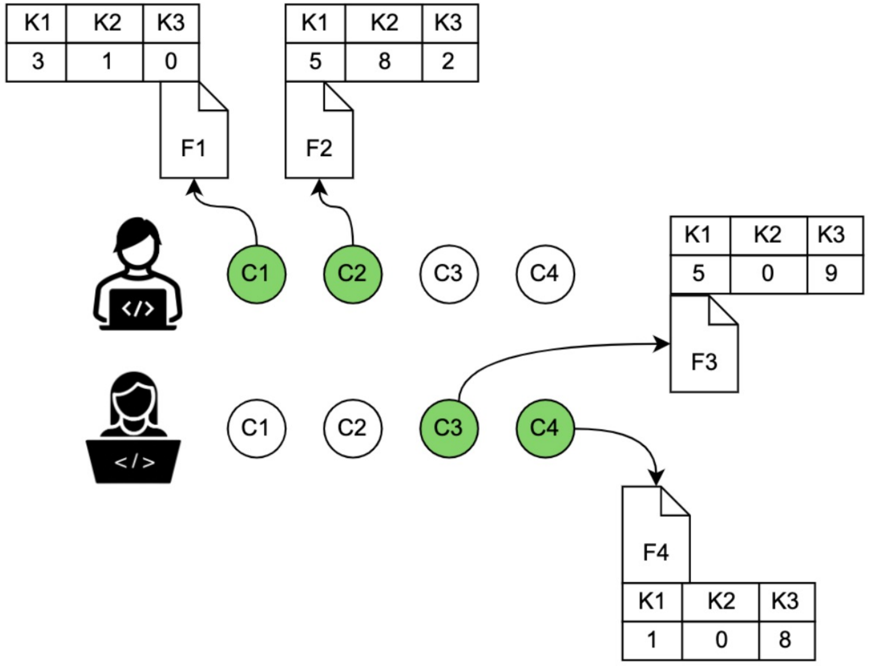


(a) Developers' commit activity and the changed files

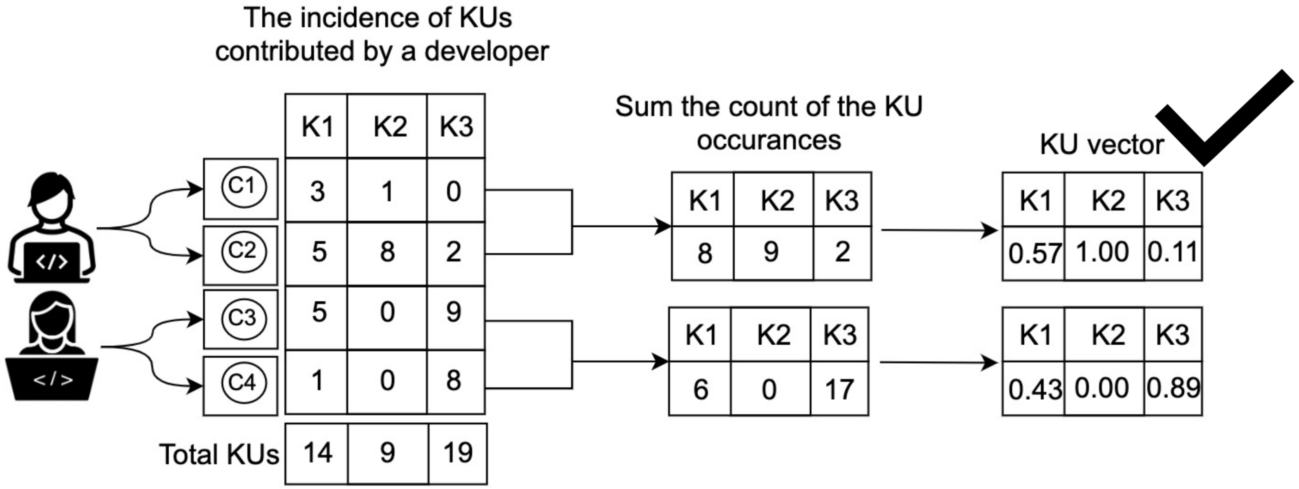


(b) Representation of developer's expertise with KUs

We represent developers’ expertise with KUs that are associated with changed files in commits



(a) Developers' commit activity and the changed files



(b) Representation of developer’s expertise with KUs

We collected **290k** commit data and **65k** pull request data from 8 active Java projects in GitHub

APACHE ACTIVEMQ®

HIVE

Groovy

APACHE LUCENE™

APACHE STORM™

APACHE HBASE

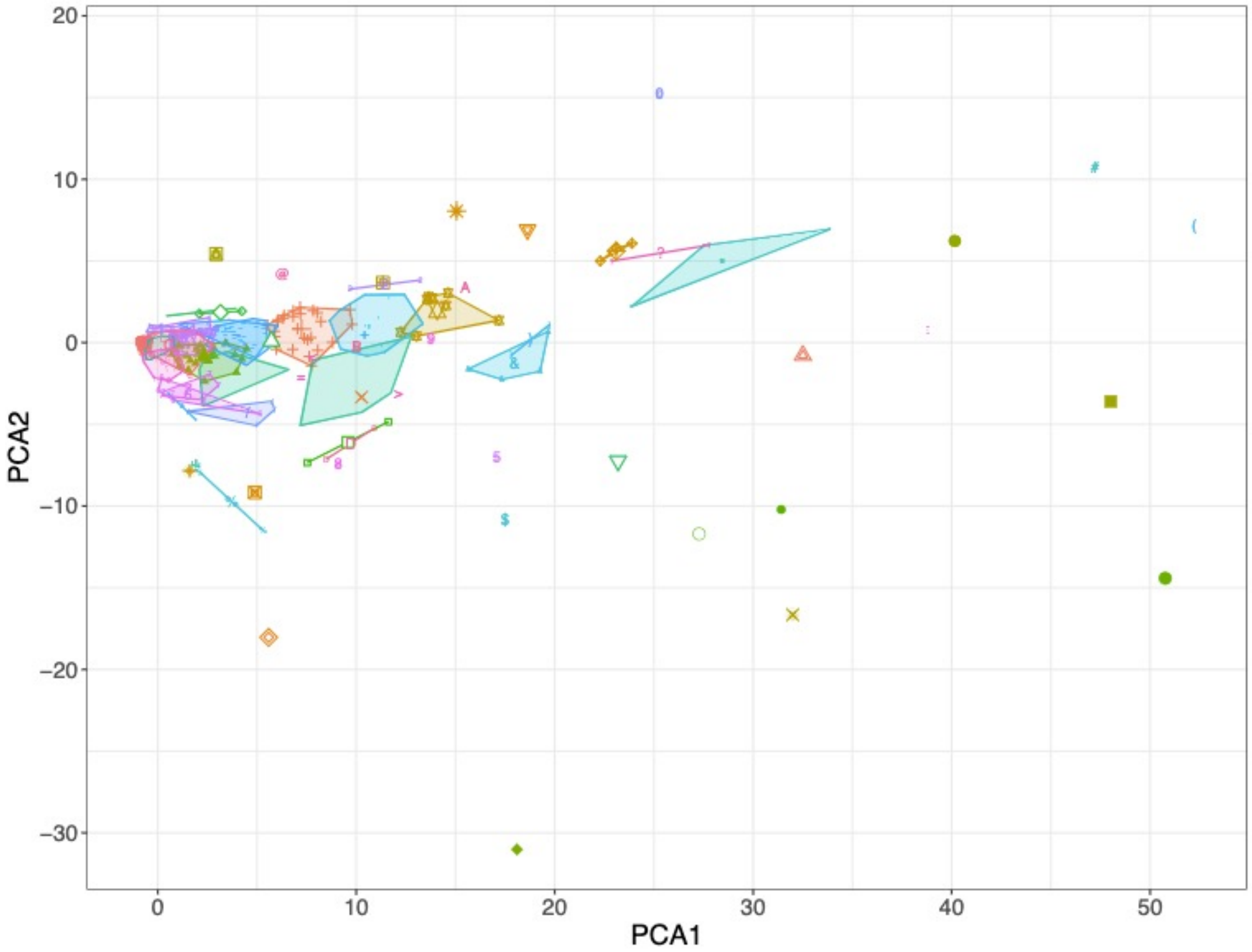
APACHE WICKET™

elasticsearch

Preliminary Study: Do KUs provide a new lens to study developers’ expertise?

KUs offer a fine-grained lens to study developers' expertise

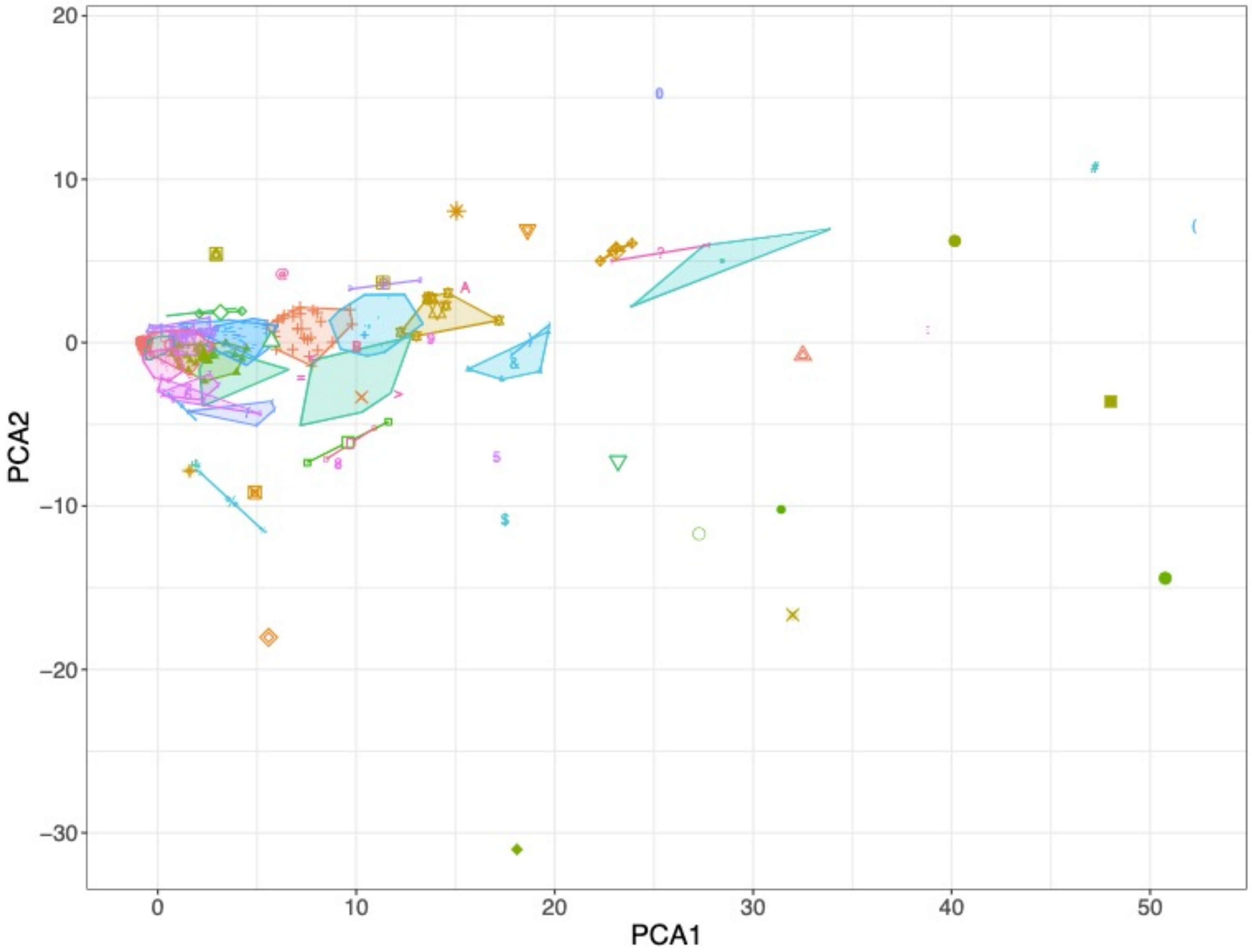
KUs offer a fine-grained lens to study developers' expertise



Developer clusters with KU-based expertise profile

KUs offer a fine-grained lens to study developers' expertise

KUs identify **71** different clusters of developers

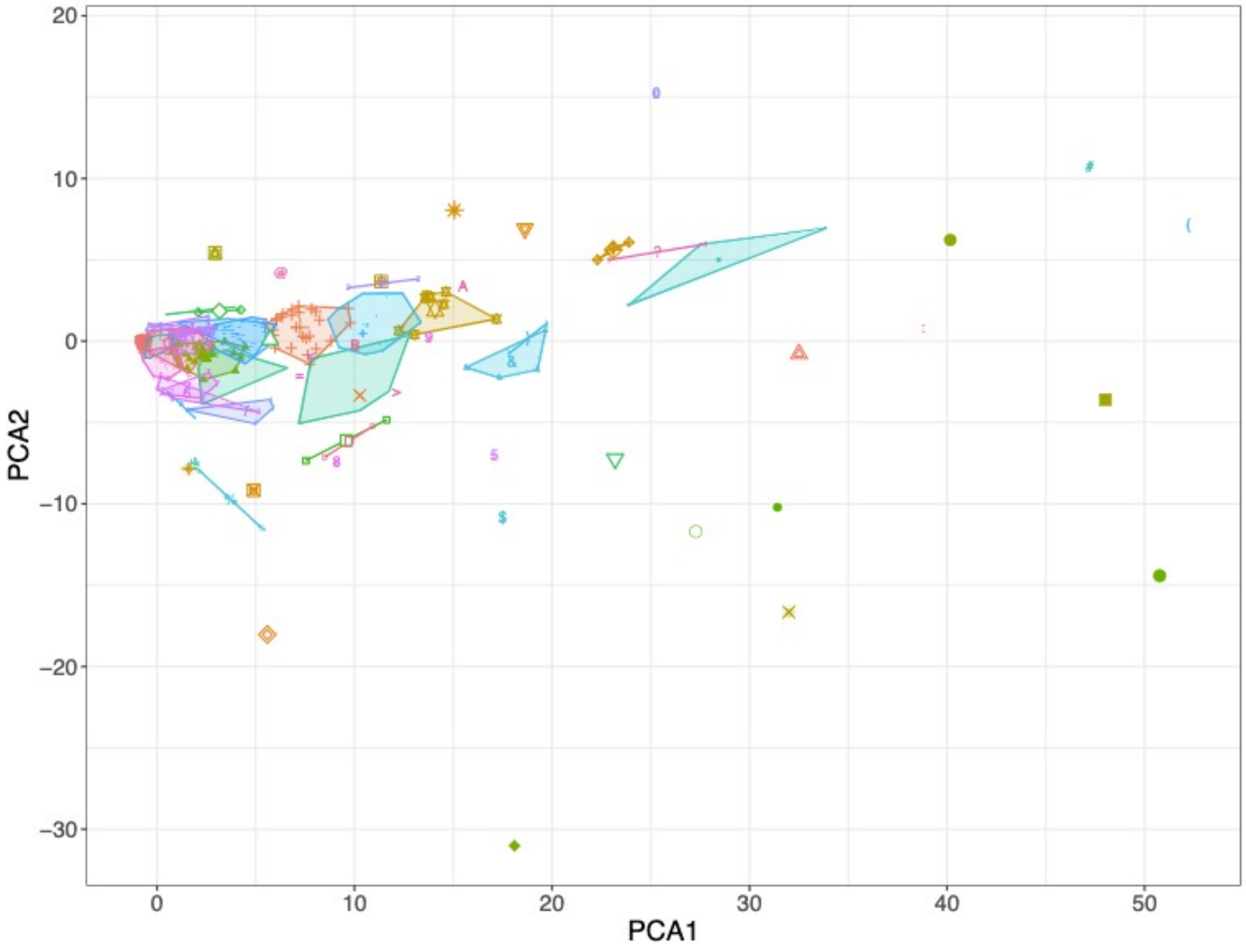


Developer clusters with KU-based expertise profile

KUs offer a fine-grained lens to study developers' expertise

KUs identify **71** different clusters of developers

The **Gini index** for clusters' size is **0.96**, indicating that there exists a very **strong inequality**

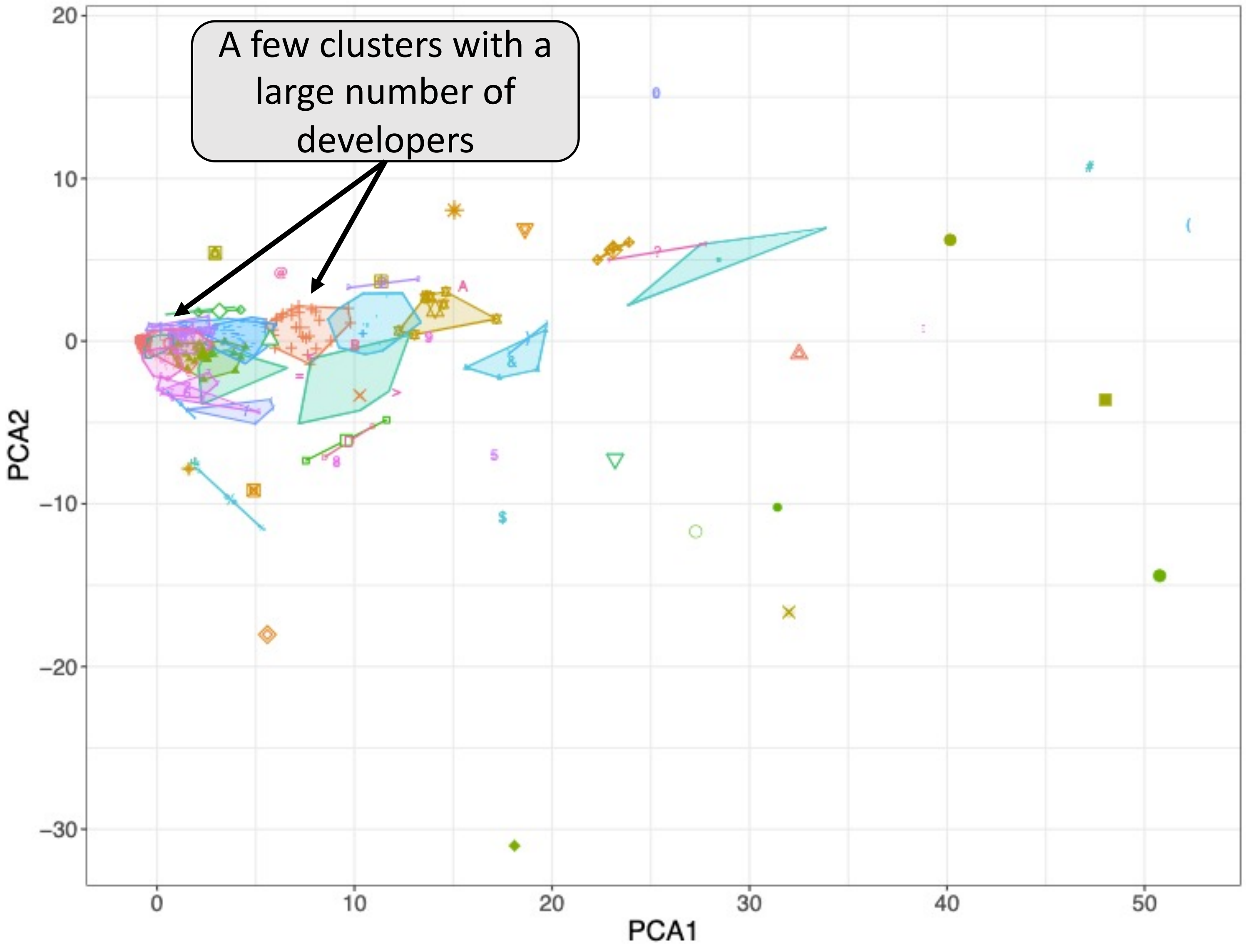


Developer clusters with KU-based expertise profile

KUs offer a fine-grained lens to study developers' expertise

KUs identify **71** different clusters of developers

The **Gini index** for clusters' size is **0.96**, indicating that there exists a very **strong inequality**

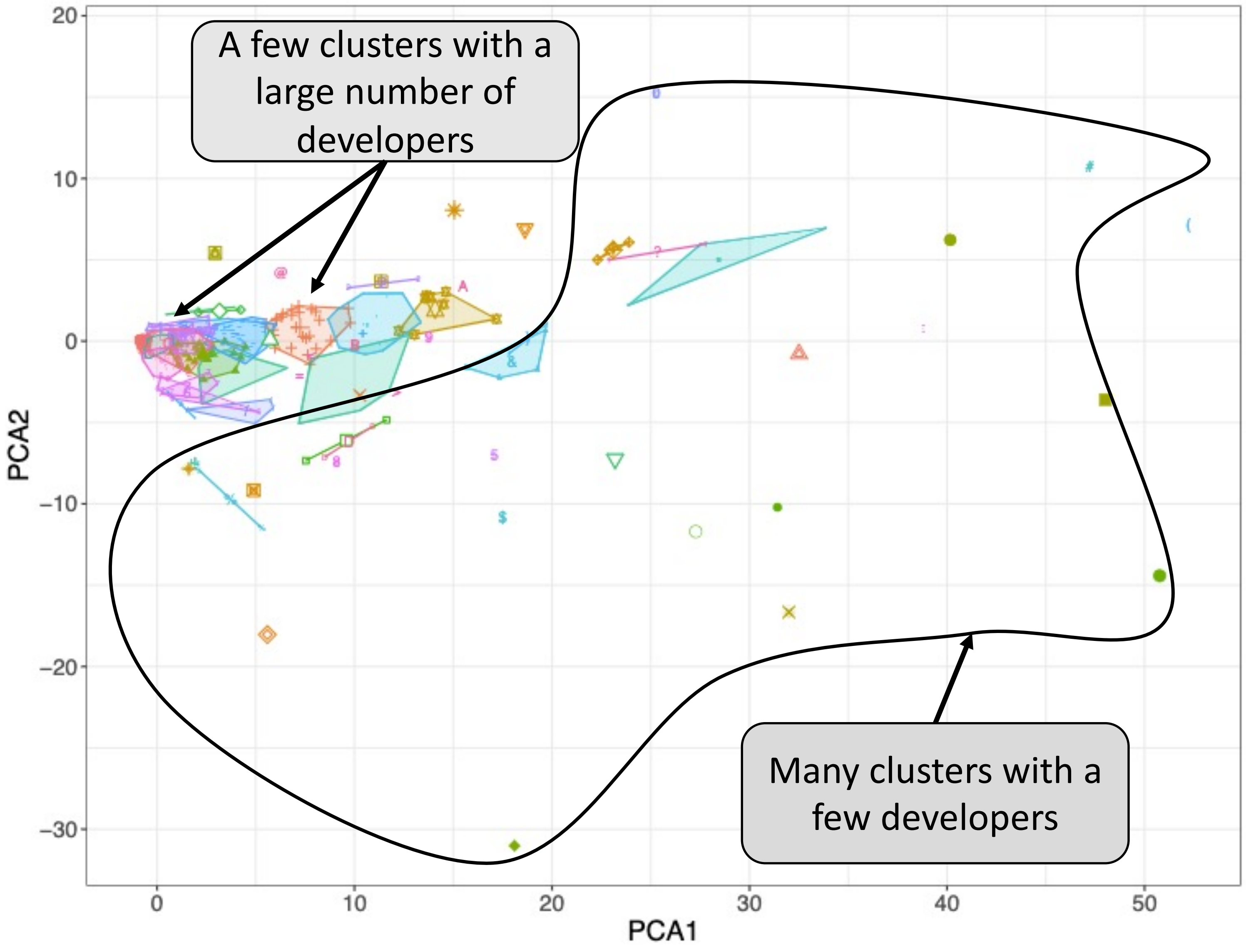


Developer clusters with KU-based expertise profile

KUs offer a fine-grained lens to study developers' expertise

KUs identify **71** different clusters of developers

The **Gini index** for clusters' size is **0.96**, indicating that there exists a very **strong inequality**



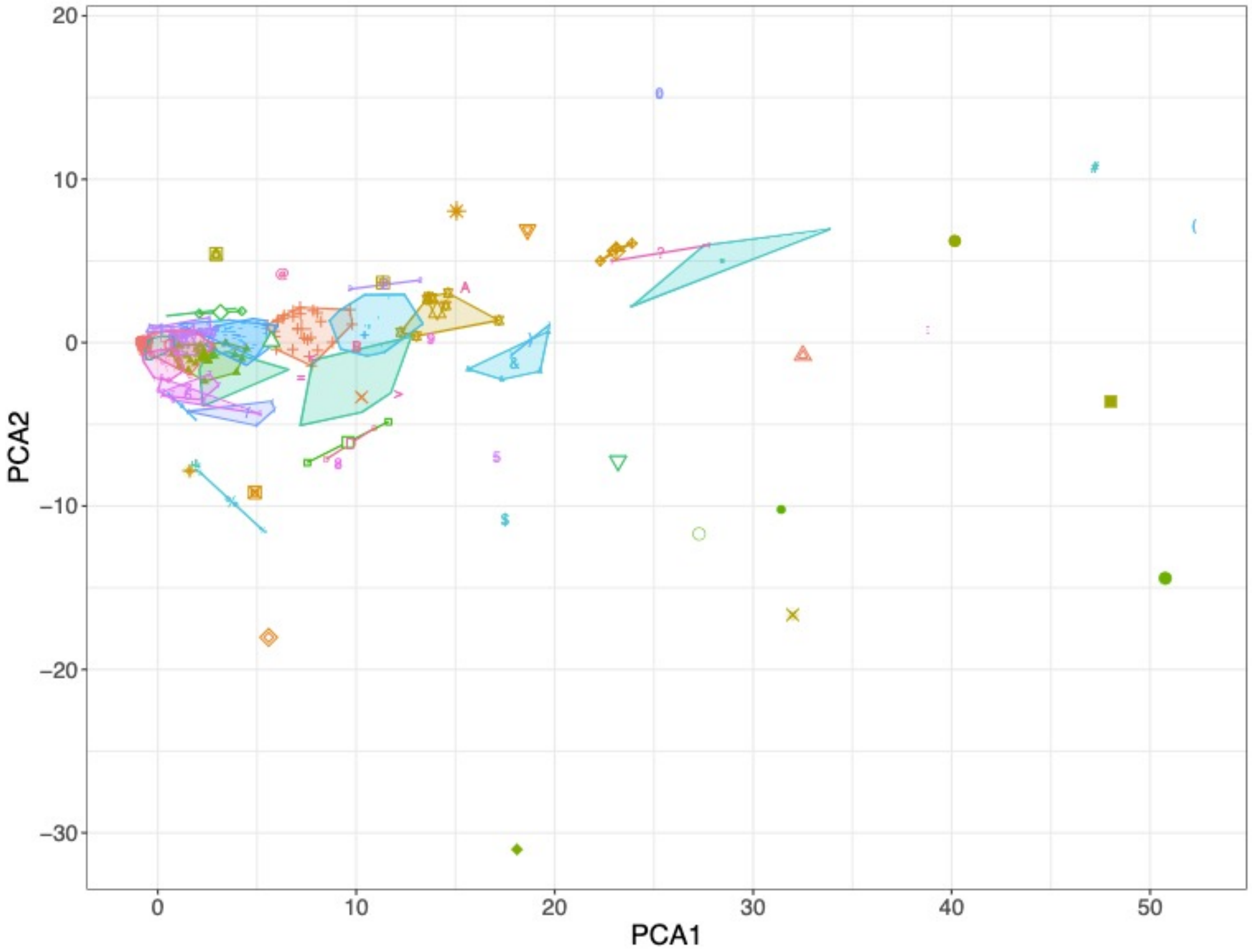
Developer clusters with KU-based expertise profile

KUs offer a fine-grained lens to study developers' expertise

KUs identify **71** different clusters of developers

The **Gini index** for clusters' size is **0.96**, indicating that there exists a very **strong inequality**

57% of the generated clusters are **singleton** (i.e., the size of these clusters is one)



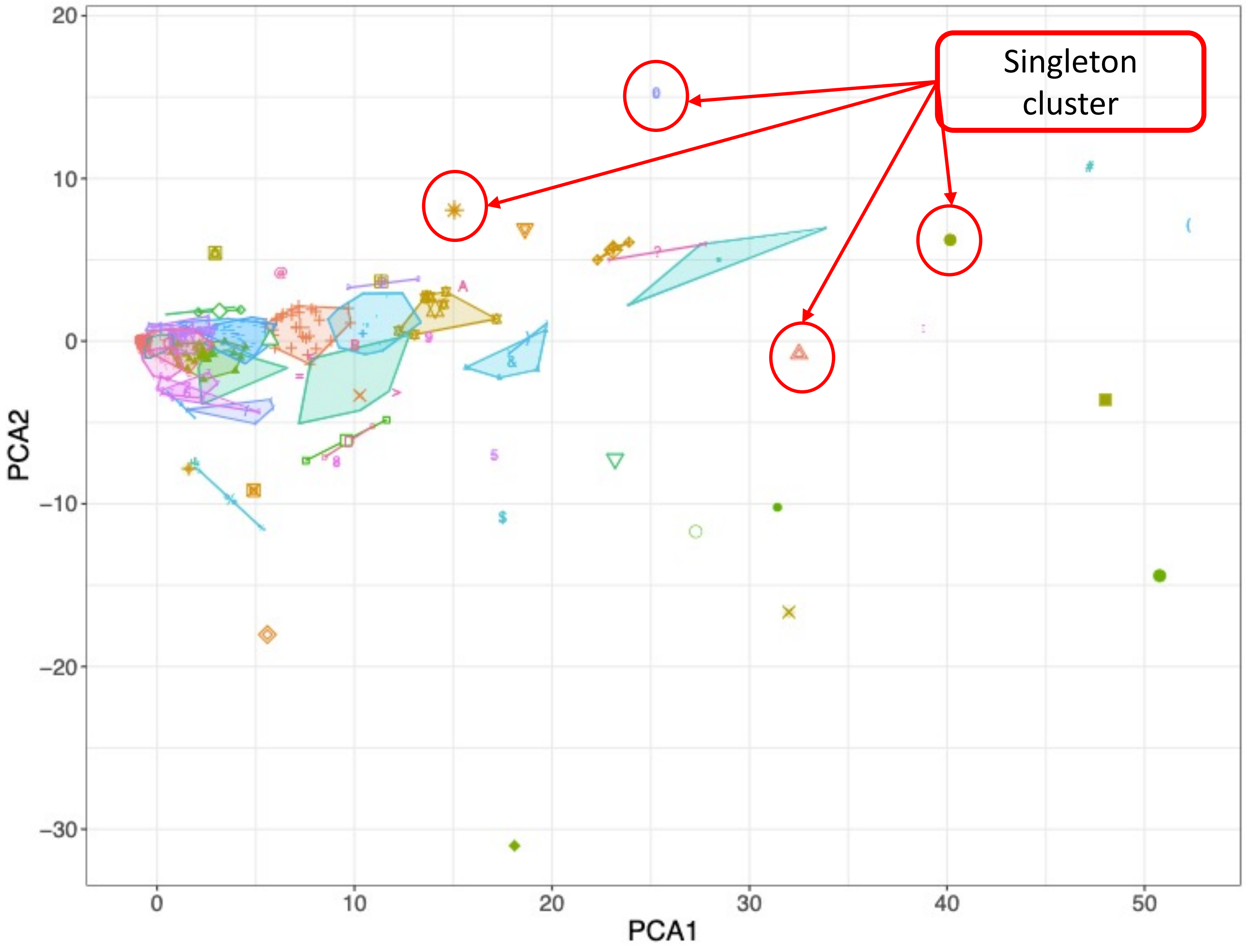
Developer clusters with KU-based expertise profile

KUs offer a fine-grained lens to study developers' expertise

KUs identify **71** different clusters of developers

The **Gini index** for clusters' size is **0.96**, indicating that there exists a very **strong inequality**

57% of the generated clusters are **singleton** (i.e., the size of these clusters is one)



Developer clusters with KU-based expertise profile

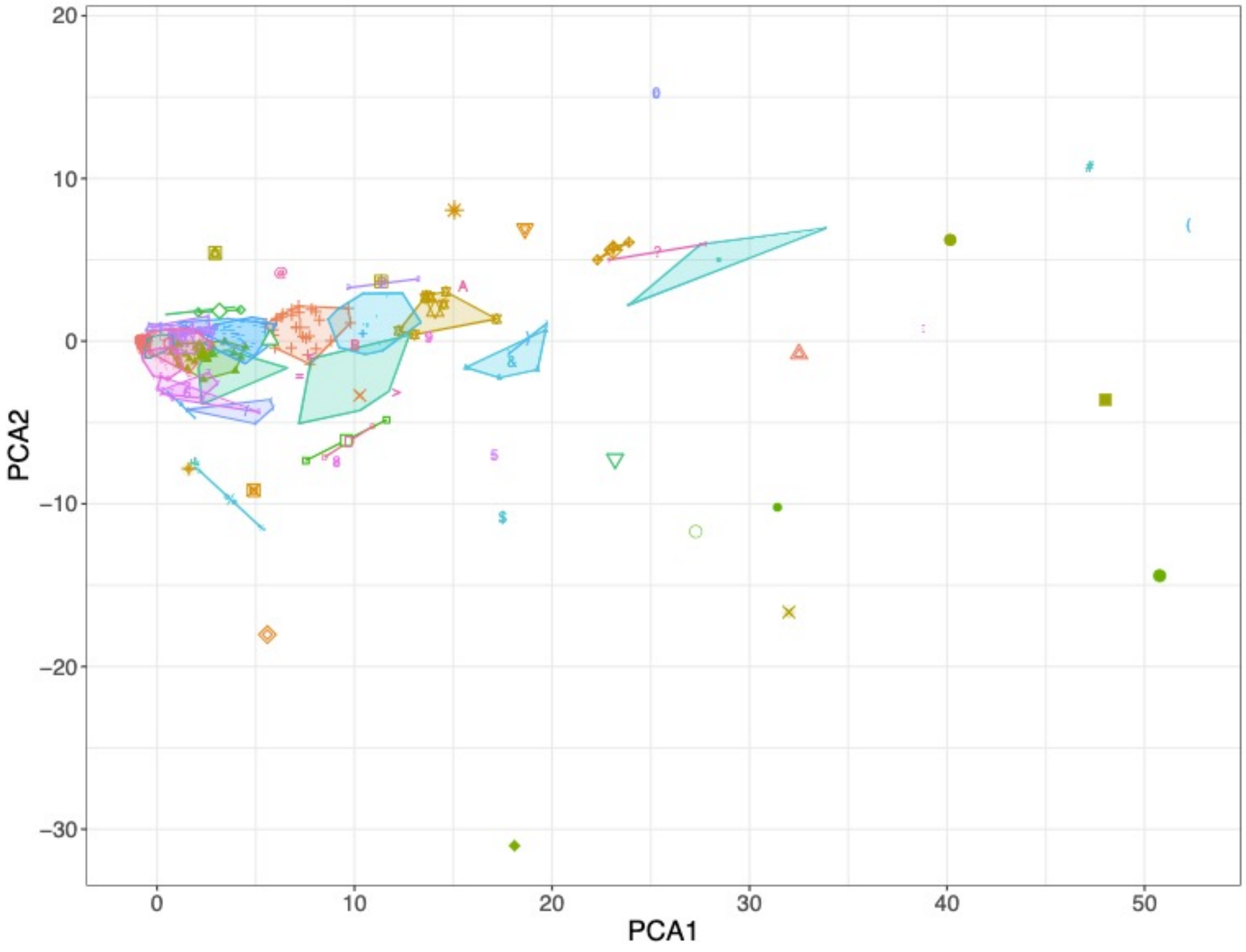
KUs offer a fine-grained lens to study developers' expertise

KUs identify **71** different clusters of developers

The **Gini index** for clusters' size is **0.96**, indicating that there exists a very **strong inequality**

57% of the generated clusters are **singleton** (i.e., the size of these clusters is one)

Each cluster hosts a set of developers with **unique KU-based expertise profile**



Developer clusters with KU-based expertise profile

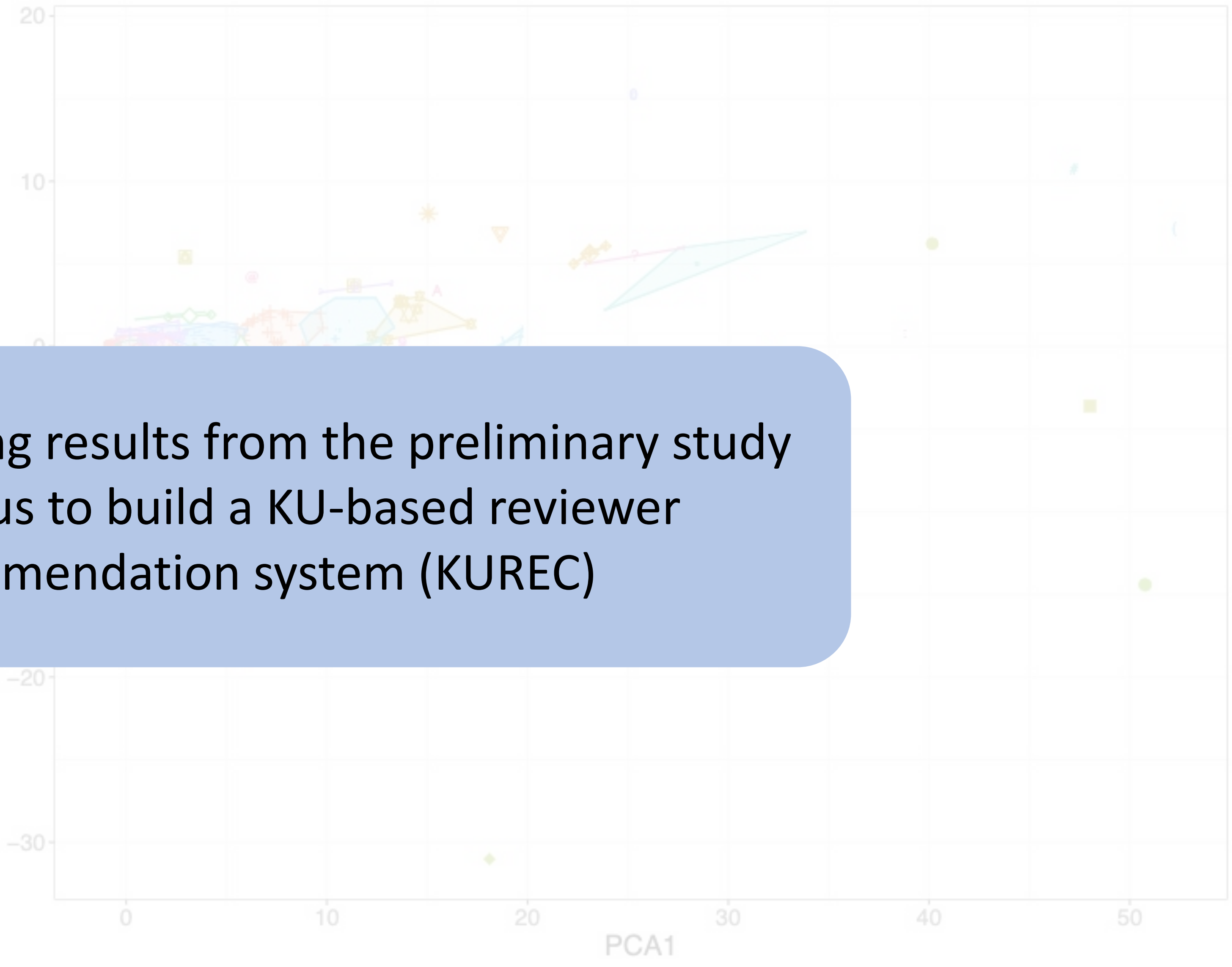
KUs offer a fine-grained lens to study developers' expertise

KUs identify **71** different clusters of developers

The **Gini index** for clusters' size is **0.96**, indicating that there exists a

the size of these clusters is one)

Each cluster hosts a set of developers with **unique KU-based expertise profile**



Developer clusters with KU-based expertise profile

Our encouraging results from the preliminary study motivate us to build a KU-based reviewer recommendation system (KUREC)

We address three research questions (RQs)

RQ1: How accurately can KUREC recommend code reviewers in pull requests?

RQ2: Can KUREC be made more accurate by combining it with existing recommenders?

RQ3: How reasonable are those recommendations of KUREC that are not matched with ground truth data?

We address three research questions (RQs)

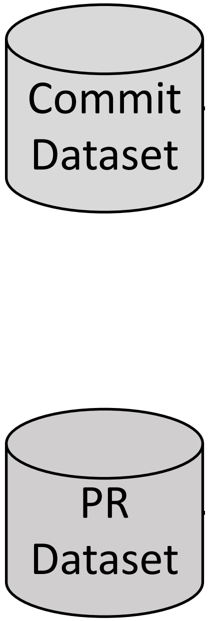
RQ1: How accurately can KUREC recommend code reviewers in pull requests?

RQ2: Can KUREC be made more accurate by combining it with existing recommenders?

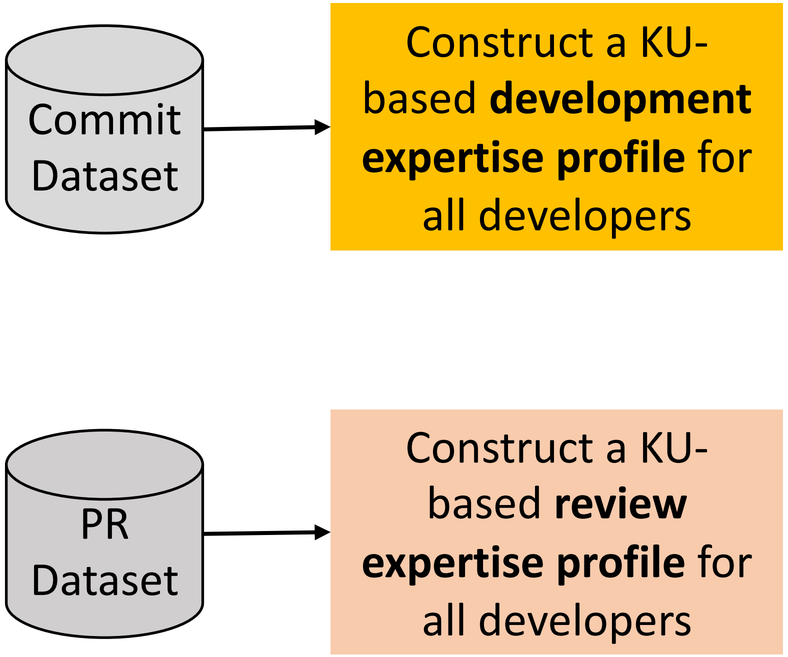
RQ3: How reasonable are those recommendations of KUREC that are not matched with ground truth data?

Our approach for building KUREC recommender

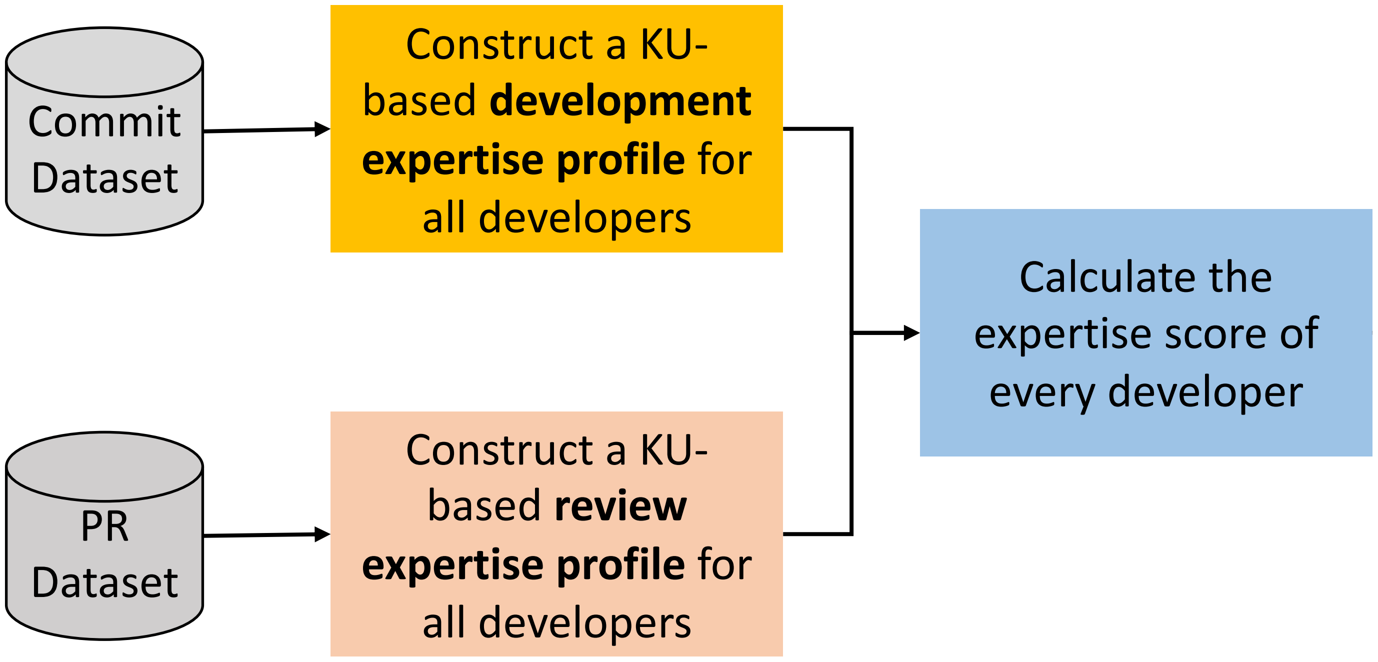
Our approach for building KUREC recommender



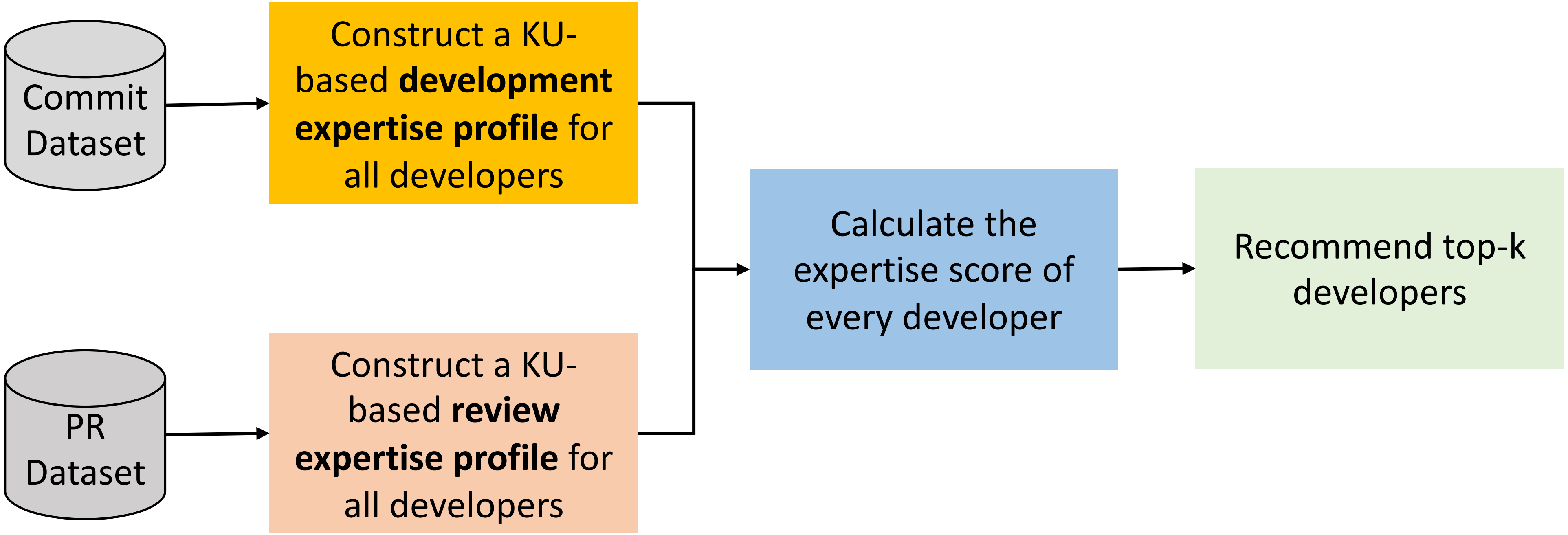
Our approach for building KUREC recommender



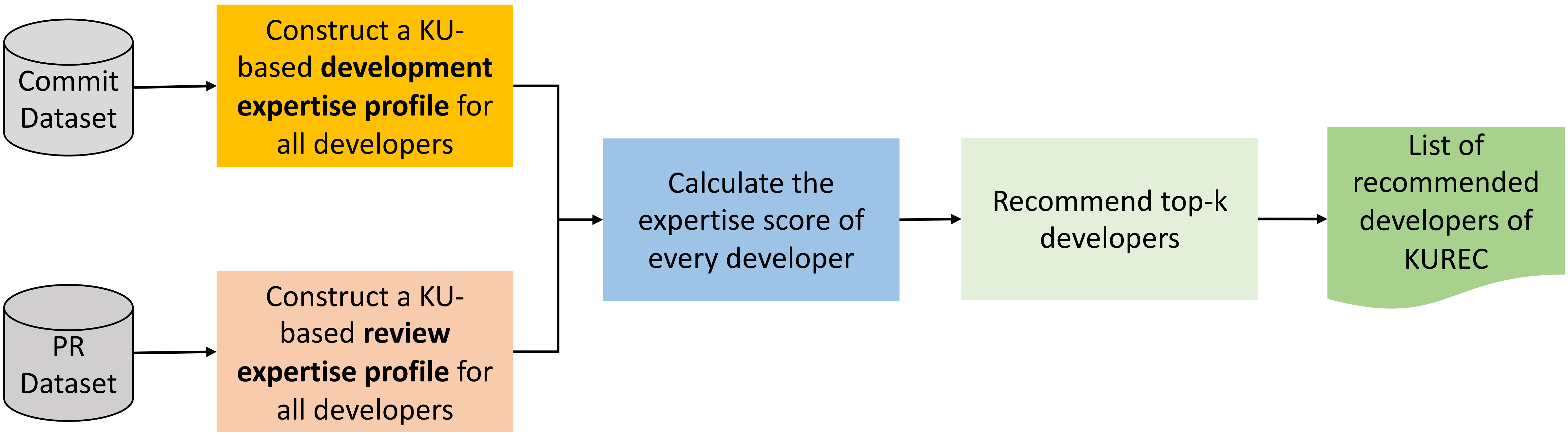
Our approach for building KUREC recommender



Our approach for building KUREC recommender



Our approach for building KUREC recommender



We construct four baseline recommenders

We construct four baseline recommenders

[1] Commit-frequency-based recommender (CF) (MSR 2013)

The recommender sorts developers in decreasing order of commit counts and recommends the top-k ones

We construct four baseline recommenders

[1] Commit-frequency-based recommender (CF)
(MSR 2013)

The recommender sorts developers in decreasing order of commit counts and recommends the top-k ones

[2] Review-frequency-based recommender (RF)
(APSEC 2014)

The recommender sorts developers in decreasing order of review counts and recommends the top-k ones

We construct four baseline recommenders

[1] Commit-frequency-based recommender (CF) (MSR 2013)

The recommender sorts developers in decreasing order of commit counts and recommends the top-k ones

[2] Review-frequency-based recommender (RF) (APSEC 2014)

The recommender sorts developers in decreasing order of review counts and recommends the top-k ones

[3] Modification-expertise-based recommender (ER) (CCSC 2000)

The recommender sorts developers in reverse chronological order based on the date who last modified the changed file in a given PR. Finally, ER recommends the top-k ranked developers

We construct four baseline recommenders

[1] Commit-frequency-based recommender (CF) (MSR 2013)

The recommender sorts developers in decreasing order of commit counts and recommends the top-k ones

[2] Review-frequency-based recommender (RF) (APSEC 2014)

The recommender sorts developers in decreasing order of review counts and recommends the top-k ones

[3] Modification-expertise-based recommender (ER) (CCSC 2000)

The recommender sorts developers in reverse chronological order based on the date who last modified the changed file in a given PR. Finally, ER recommends the top-k ranked developers

[4] Review-history-based recommender (CHREV) (TSE 2016)

CHREV distills review contribution into three measures:

(1) total number of review comments
(2) total number of workdays
(3) recency of the review comments

CHREV generates a score for every developer based on these measures, sorts developers decreasing order of the score and recommends top-k ones

We use two popular metrics to evaluate the performance of recommenders

We use two popular metrics to evaluate the performance of recommenders

Top-k Accuracy

$$Top\text{-}k\ accuracy = \frac{\sum_{r \in R} isCorrect(r, Top-k)}{|R|}$$

Here, R denotes the set of PRs in the test dataset. The isCorrect(r, Top−k) returns 1 if at least one of top-k developers is the correct reviewer of the PR r and returns 0 otherwise.

We use two popular metrics to evaluate the performance of recommenders

Top-k Accuracy

$$Top\text{-}k\ accuracy = \frac{\sum_{r \in R} isCorrect(r, Top-k)}{|R|}$$

Here, R denotes the set of PRs in the test dataset. The isCorrect(r, Top−k) returns 1 if at least one of top-k developers is the correct reviewer of the PR r and returns 0 otherwise.

Mean Average Precision (MAP)

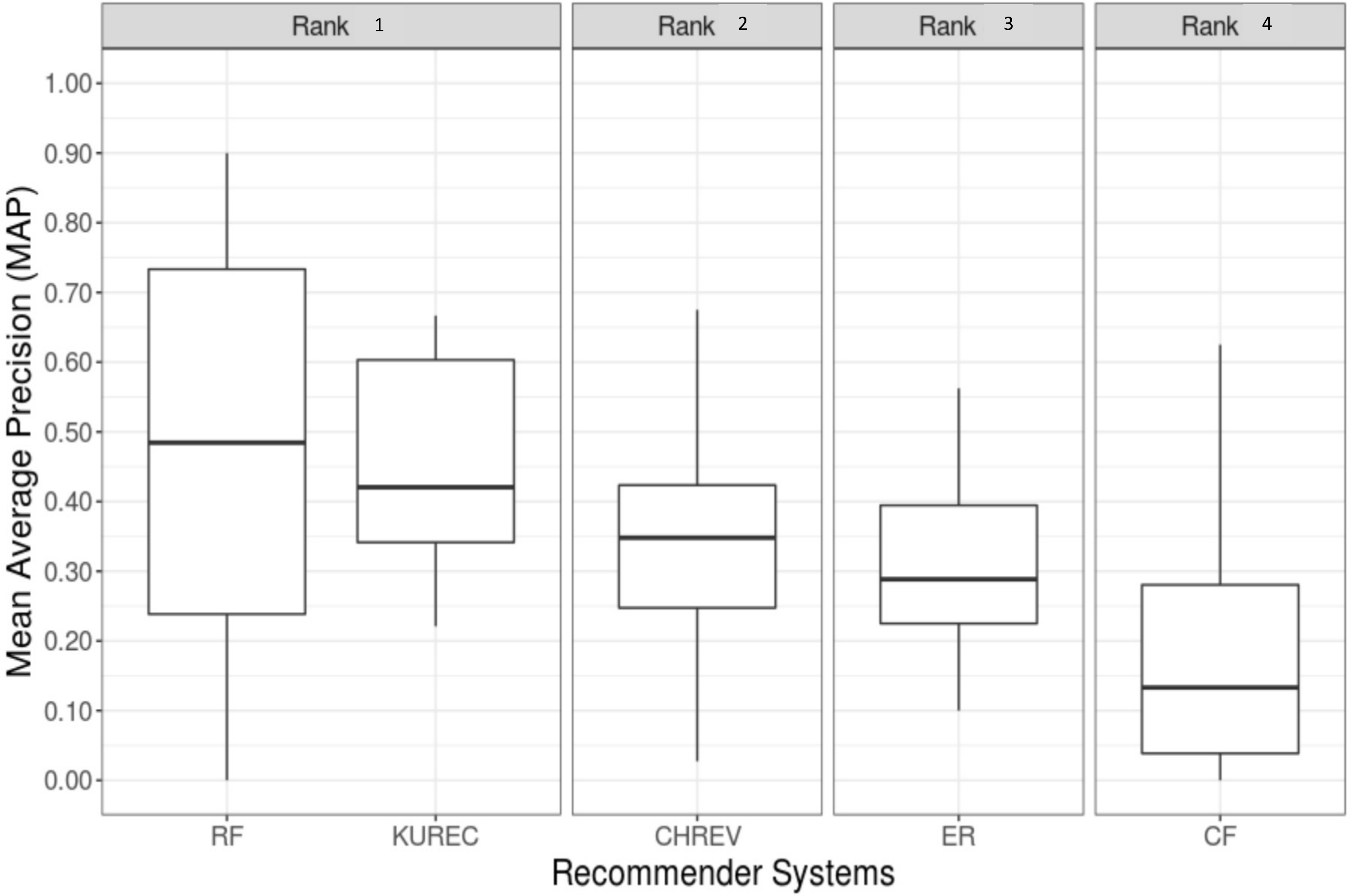
$$AP@k = \frac{\sum_{i=1}^{k} \frac{s(i)}{i} \times rel(i)}{\sum_{i=1}^{k} rel(i)}$$

MAP @k is the average of AP@k over all the PRs in the test dataset

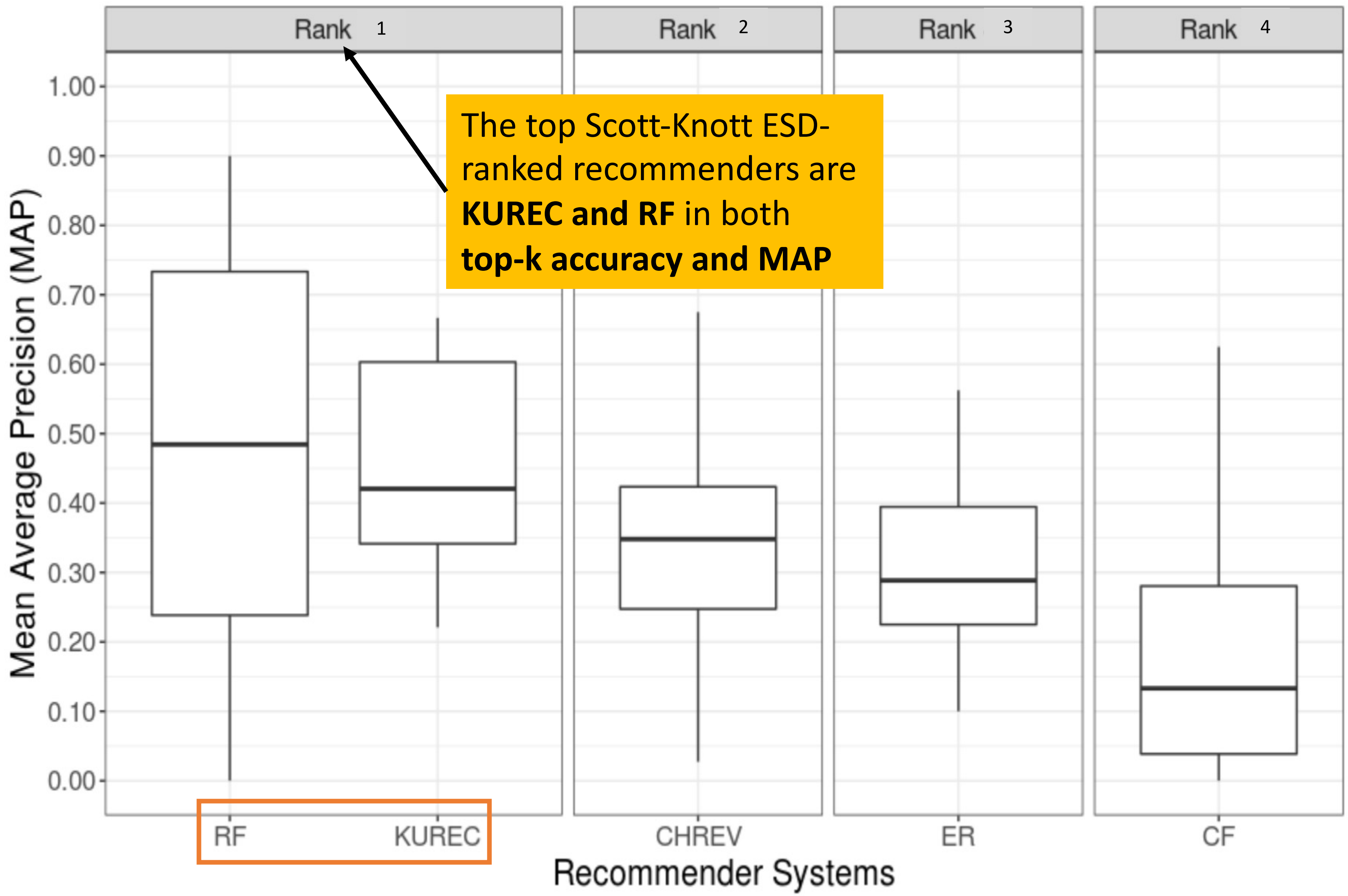
Here, i is the position of each developer in the recommended list of developers, and s(i) is the sequence number of the correct developer at position i.
The rel(i) returns 1 if the ith developer in the list is correct and 0 otherwise.

KUREC is more stable than RF and outperforms the remaining three baselines

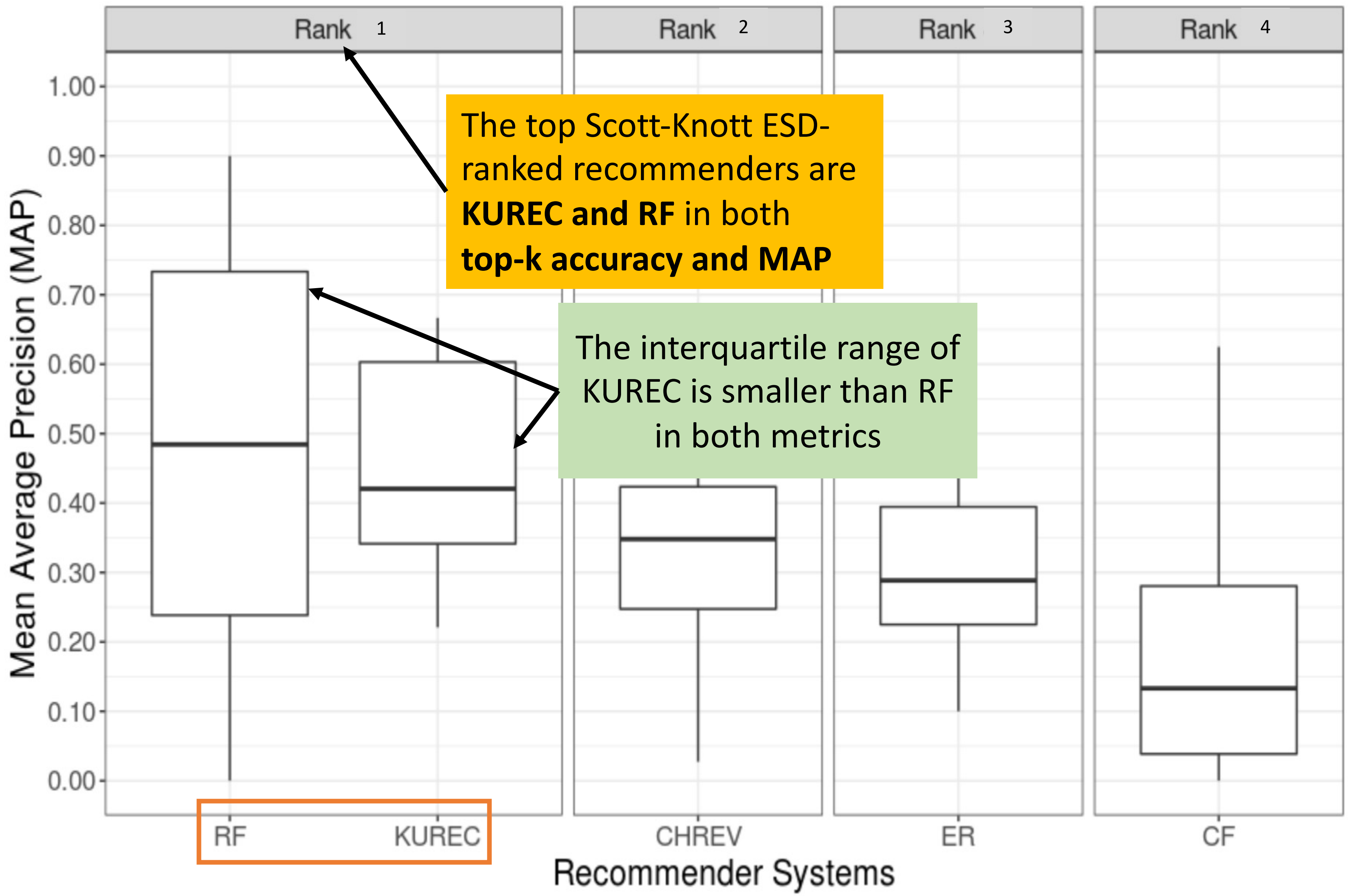
KUREC is more stable than RF and outperforms the remaining three baselines



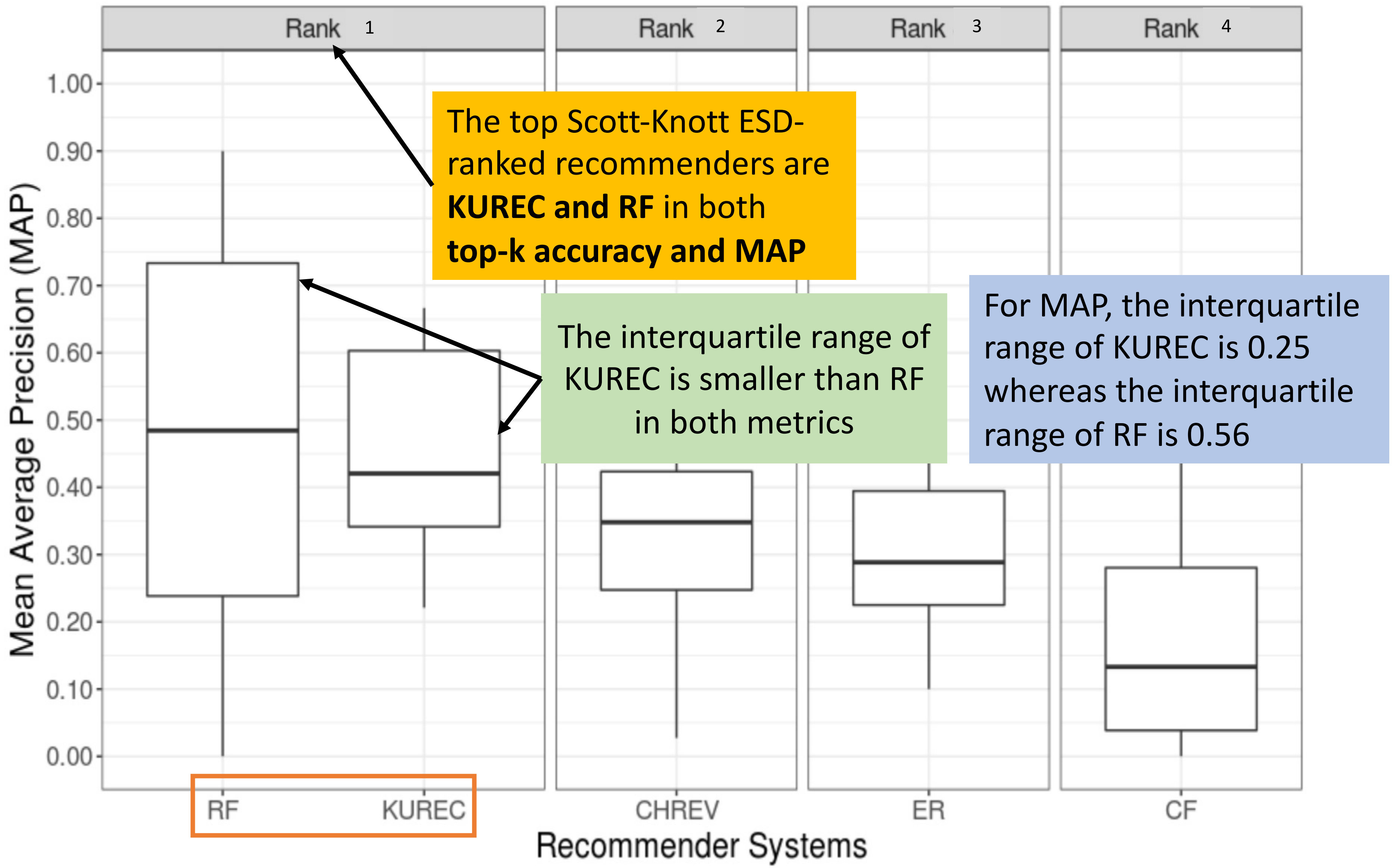
KUREC is more stable than RF and outperforms the remaining three baselines



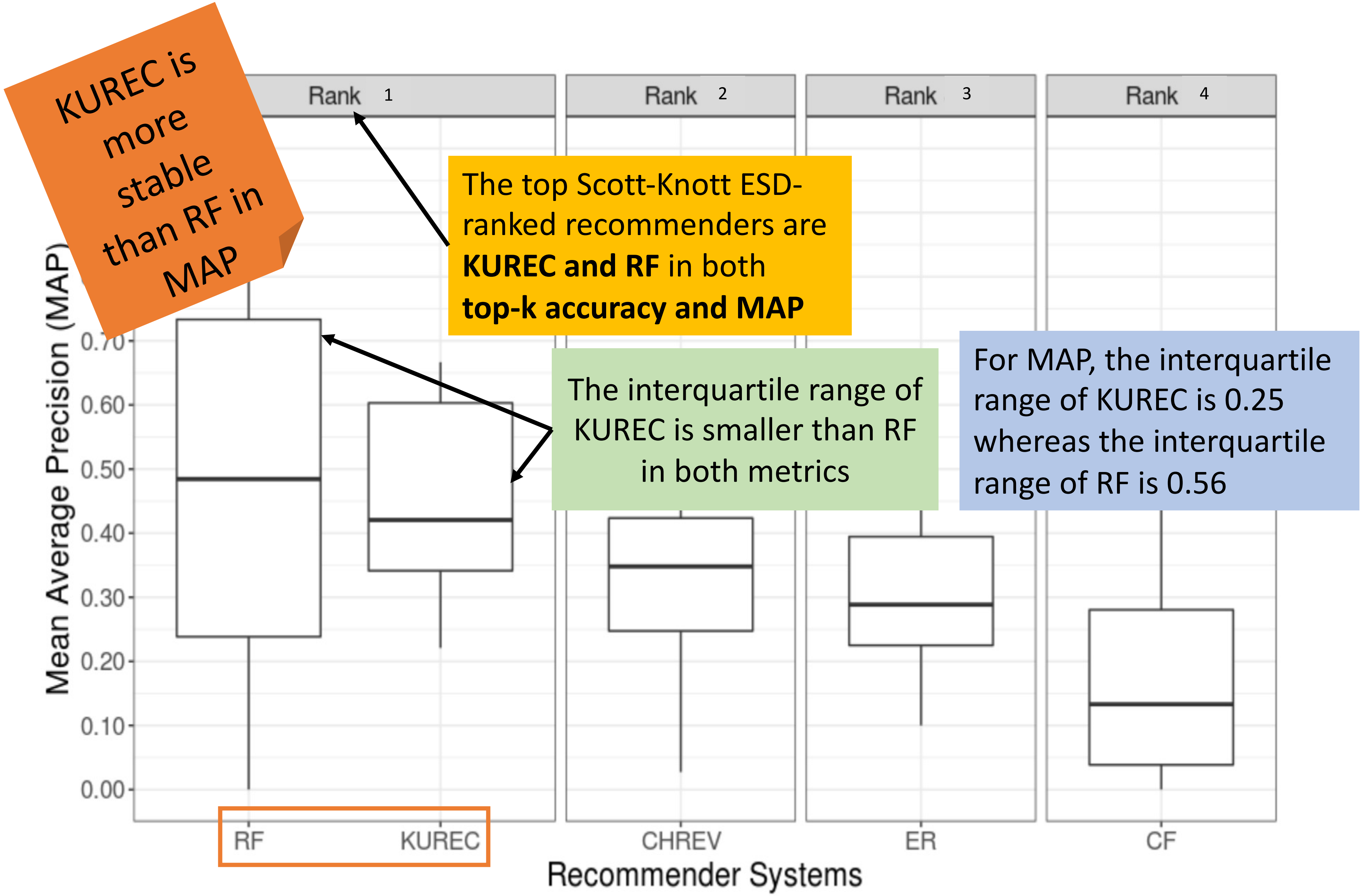
KUREC is more stable than RF and outperforms the remaining three baselines



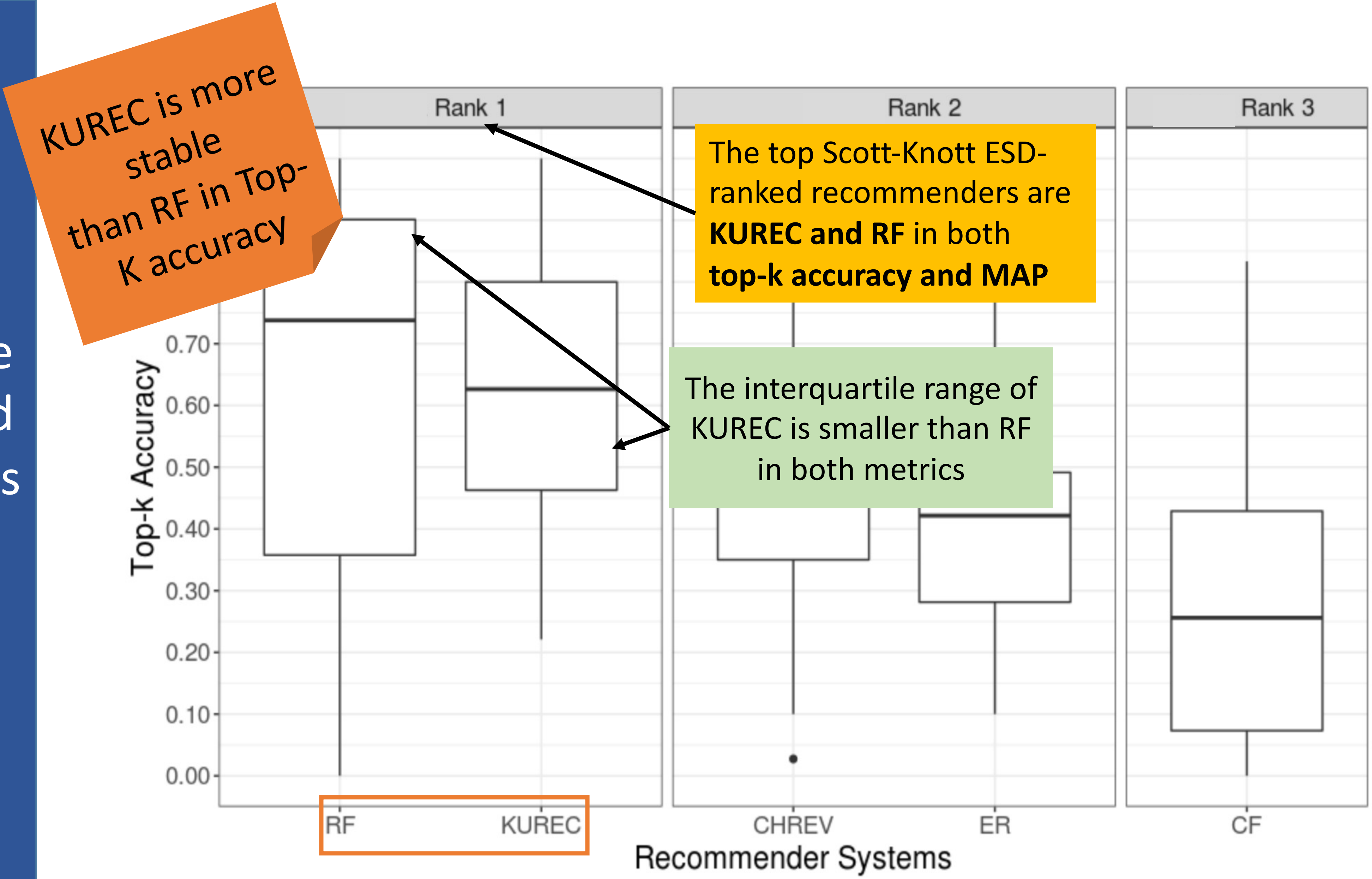
KUREC is more stable than RF and outperforms the remaining three baselines



KUREC is more stable than RF and outperforms the remaining three baselines



KUREC is more stable than RF and outperforms the remaining three baselines



Summary of RQ1

KUREC outperforms the remaining three baselines and has a more stable performance compared to RF, which is a desired property in practice

We address three research questions (RQs)

RQ1: How accurately can KUREC recommend code reviewers in pull requests?

RQ2: Can KUREC be made more accurate by combining it with existing recommenders?

RQ3: How reasonable are those recommendations of KUREC that are not matched with ground truth data?

We construct **three recommenders** by combining KUREC with the baseline recommenders

We construct **three recommenders** by combining KUREC with the baseline recommenders

To construct a combined recommender by leveraging the recommendations of different recommenders, we are **motivated by the work of Malik and Hassan [1]**

[1] Malik H, Hassan AE, "Supporting software evolution using adaptive change propagation heuristics", ICSME, 2008

We construct **three recommenders** by combining KUREC with the baseline recommenders

To construct a combined recommender by leveraging the recommendations of different recommenders, we are **motivated by the work of Malik and Hassan [1]**

In this approach, all the recommenders uses a Best Recommender System Table (BRST) to track the best-performing recommender.

[1] Malik H, Hassan AE, "Supporting software evolution using adaptive change propagation heuristics", ICSME, 2008

We construct **three recommenders** by combining KUREC with the baseline recommenders

To construct a combined recommender by leveraging the recommendations of different recommenders, we are **motivated by the work of Malik and Hassan [1]**

In this approach, all the recommenders uses a Best Recommender System Table (BRST) to track the best-performing recommender.

We implement three techniques to update the BRST and these are our new recommenders based on heuristics

[1] Malik H, Hassan AE, "Supporting software evolution using adaptive change propagation heuristics", ICSME, 2008

We construct **three recommenders** by combining KUREC with the baseline recommenders

(1) Adaptive Frequency Technique (AD_FREQ)

The BRST stores the frequency of each recommender that becomes the best performing recommender. The recommender with the highest count is selected for recommendation.

We construct **three recommenders** by combining KUREC with the baseline recommenders

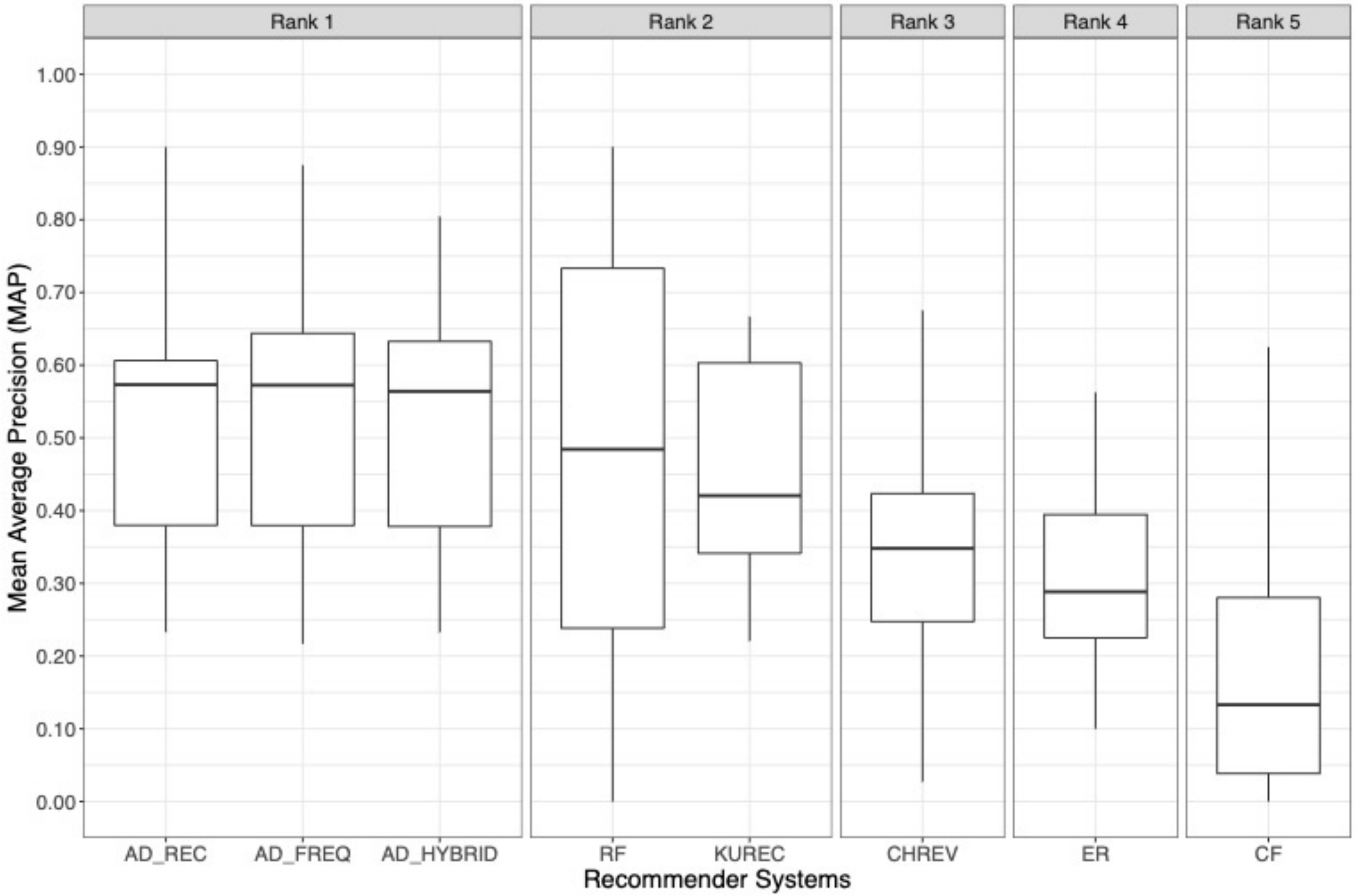
(1) Adaptive Frequency Technique (AD_FREQ)	The BRST stores the frequency of each recommender that becomes the best performing recommender. The recommender with the highest count is selected for recommendation.
(2) Adaptive Recency Technique (AD_REC)	The BRST stores the best-performing recommender that is identified in the last PR.

We construct **three recommenders** by combining KUREC with the baseline recommenders

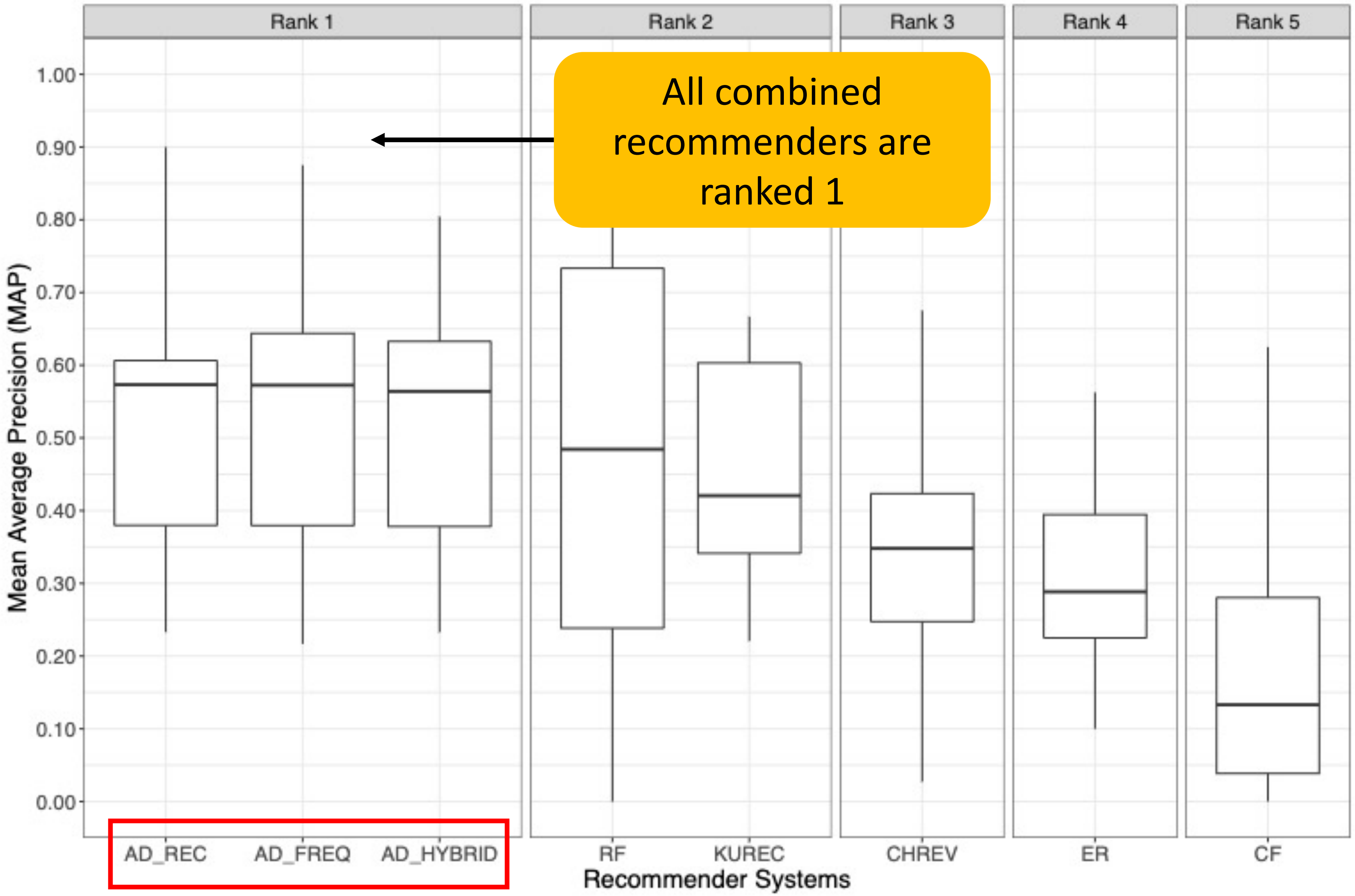
(1) Adaptive Frequency Technique (AD_FREQ)	The BRST stores the frequency of each recommender that becomes the best performing recommender. The recommender with the highest count is selected for recommendation.
(2) Adaptive Recency Technique (AD_REC)	The BRST stores the best-performing recommender that is identified in the last PR.
(3) Adaptive Hybrid Technique (AD_HYBRID)	We select the recommender that has the highest count in the BRST among the last 10 previous PRs.

All the combined recommenders outperform individual recommenders

All the combined recommenders outperform individual recommenders



All the combined recommenders outperform individual recommenders



Summary of RQ2

Combining the KU-based recommender (KUREC) with the baselines in a straight-forward manner results in better-performing recommenders

We address three research questions (RQs)

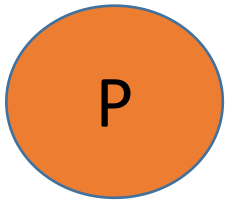
RQ1: How accurately can KUREC recommend code reviewers in pull requests?

RQ2: Can KUREC be made more accurate by combining it with existing recommenders?

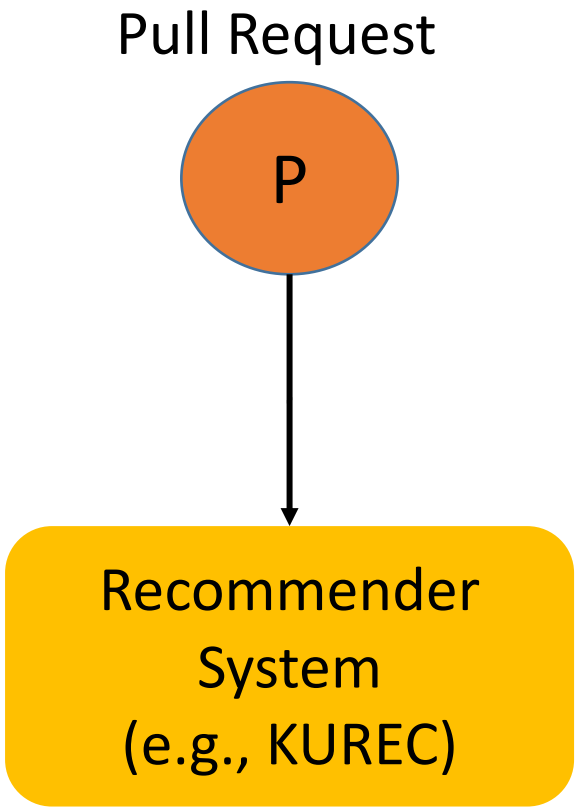
RQ3: How reasonable are those recommendations of KUREC that are not matched with ground truth data?

We study the recommendations of a recommender that does not match with ground truth data

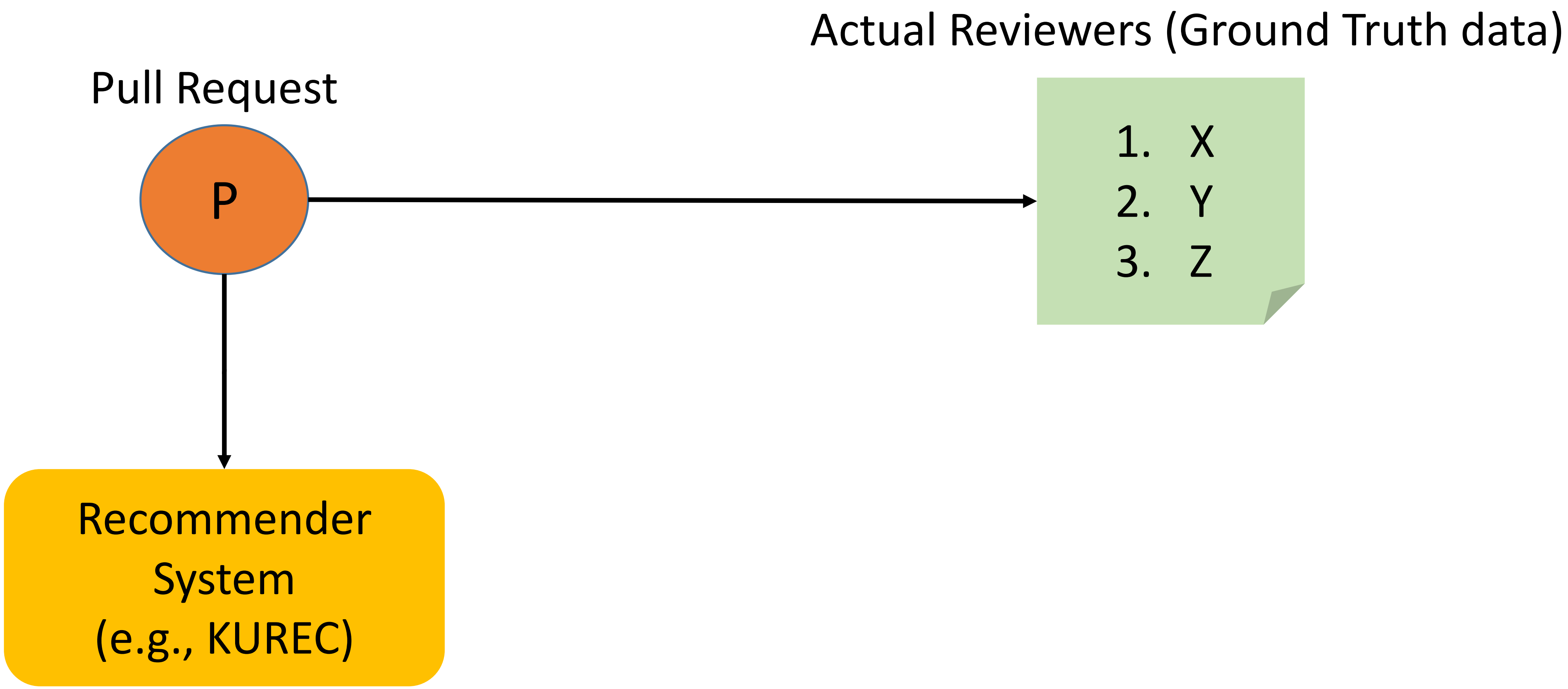
Pull Request



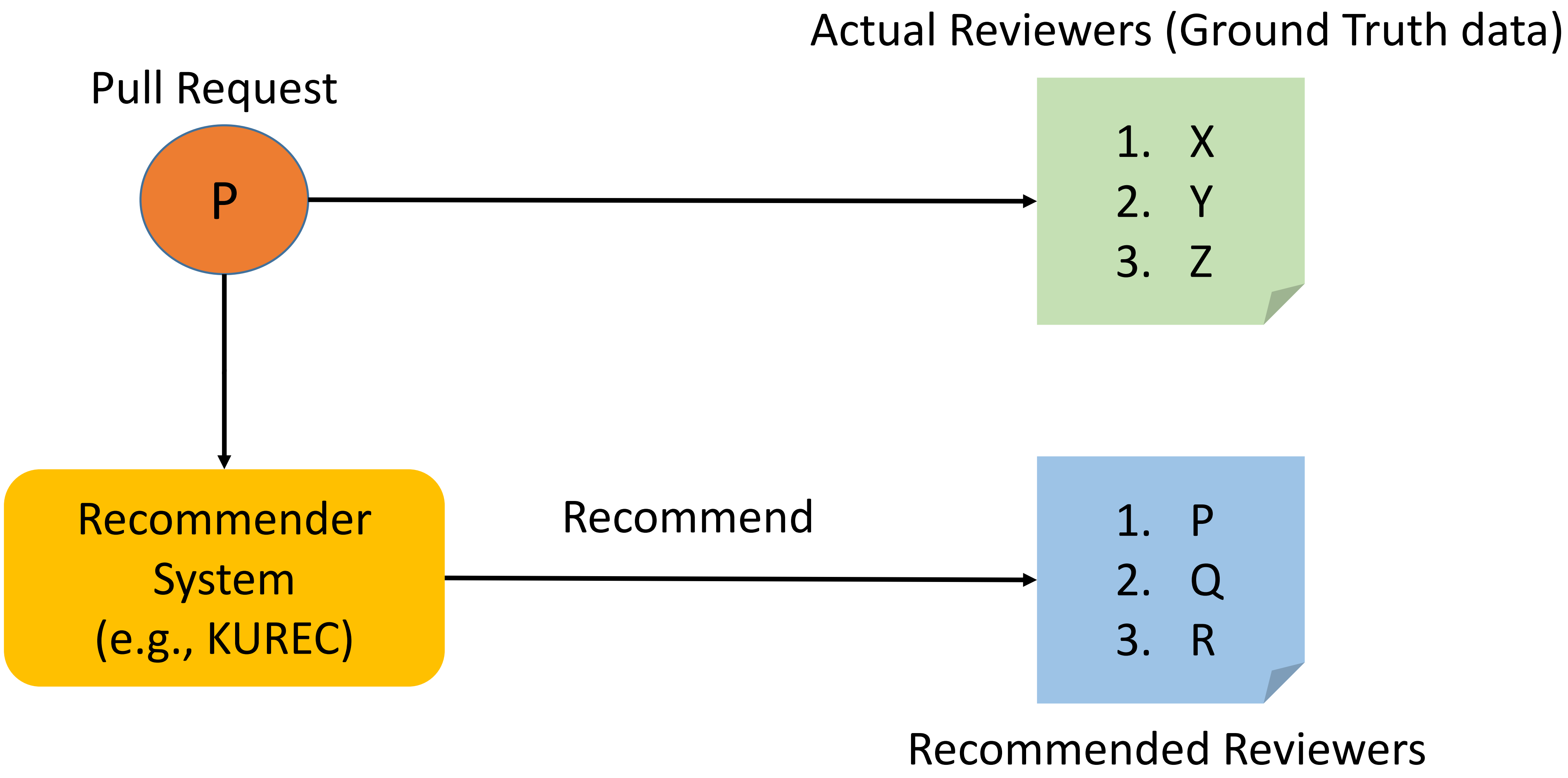
We study the recommendations of a recommender that does not match with ground truth data



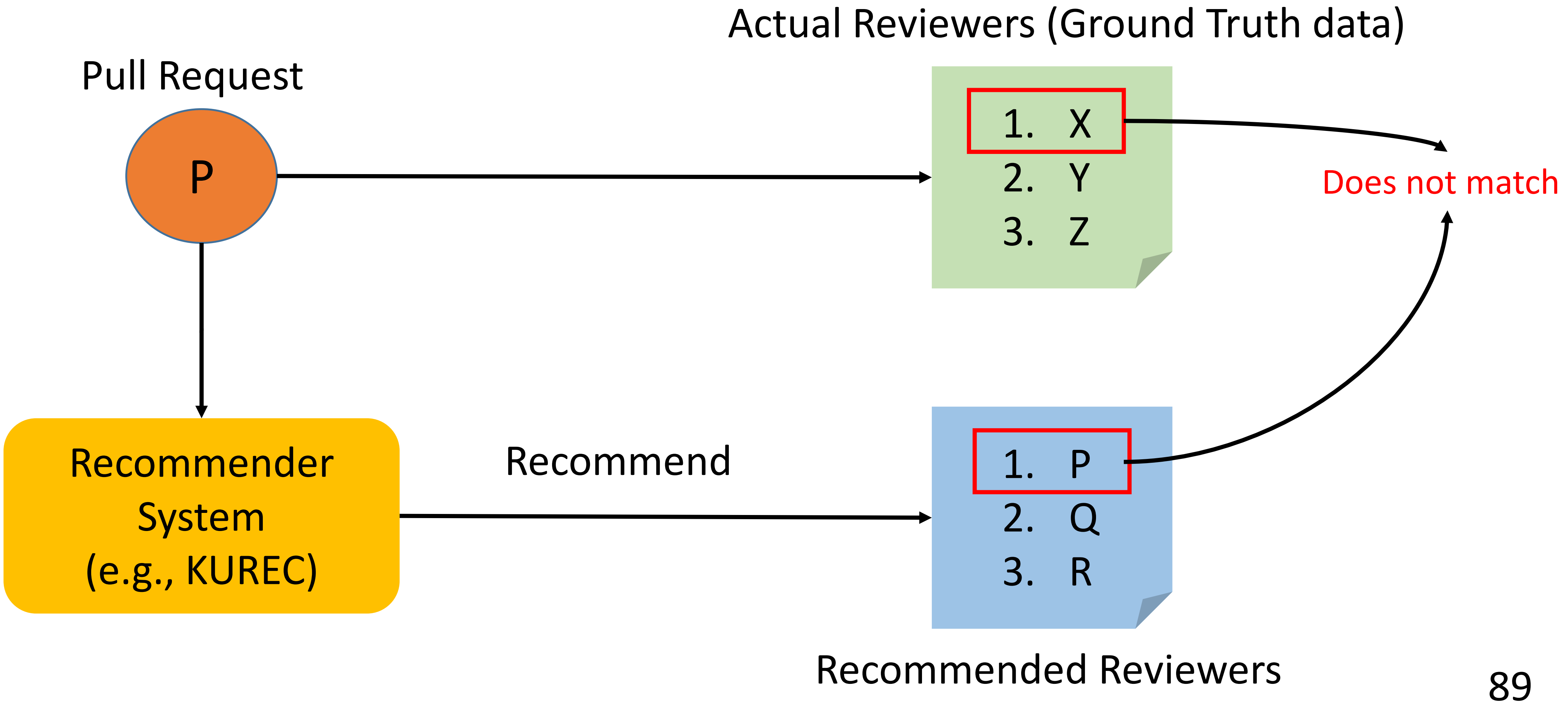
We study the recommendations of a recommender that does not match with ground truth data



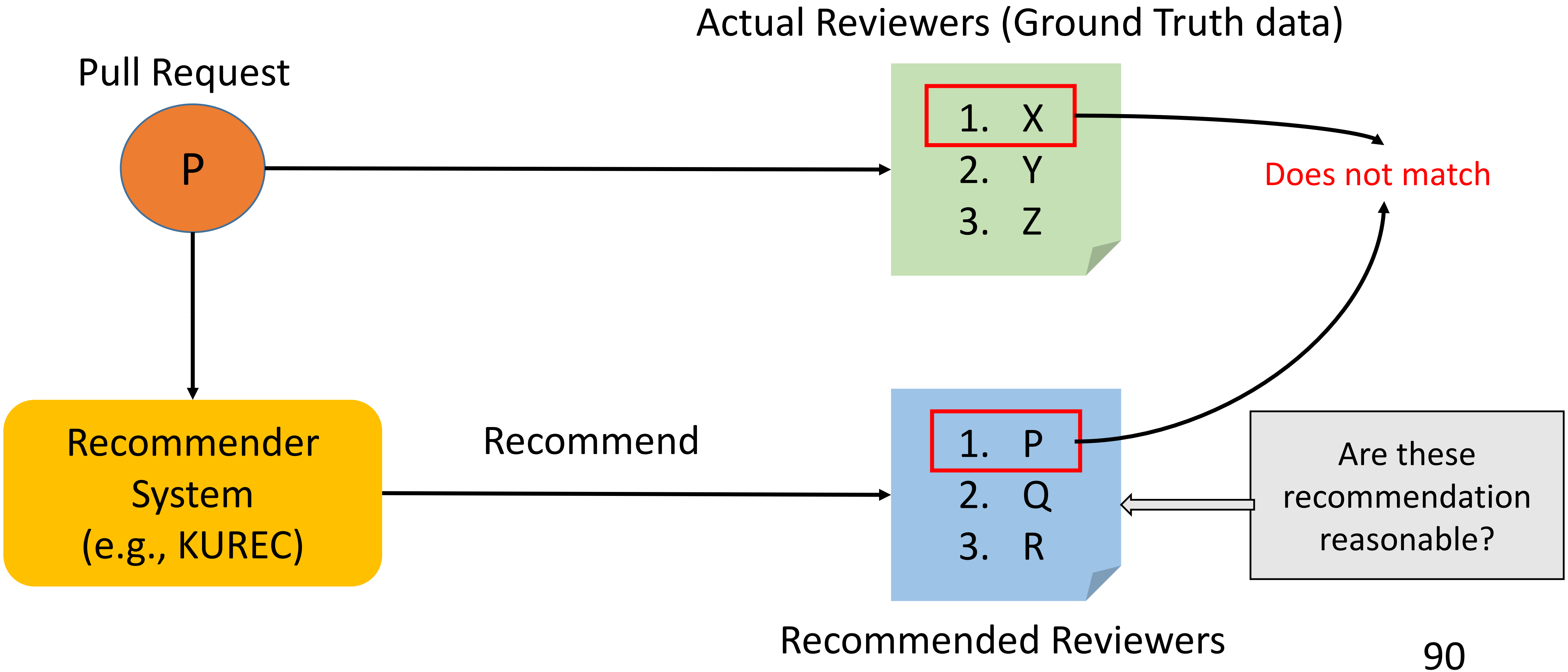
We study the recommendations of a recommender that does not match with ground truth data



We study the recommendations of a recommender that does not match with ground truth data



We study the recommendations of a recommender that does not match with ground truth data



We consider a recommendation to be **reasonable** if the recommended individual had recent (last six months) development experience with the majority (50%) of the files included in the PR in question

AD_FREQ strikes the best balance between sticking to the ground truth and reasonable recommendations

Recommender	Percentage of Reasonable recommendations
KUREC	63.4%
ER	60.9%
AD_FREQ	59.4%
AD_HYBRID	54.3%
AD_REC	54.2%
CHREV	32.7%
CF	25.4%
RF	15.2%

AD_FREQ strikes the best balance between sticking to the ground truth and reasonable recommendations

Recommender	Percentage of Reasonable recommendations
KUREC	63.4%
ER	60.9%
AD_FREQ	59.4%
AD_HYBRID	54.3%
AD_REC	54.2%
CHREV	32.7%
CF	25.4%
RF	15.2%

AD_FREQ strikes the best balance between sticking to the ground truth and reasonable recommendations

Recommender	Percentage of Reasonable recommendations
KUREC	63.4%
ER	60.9%
AD_FREQ	59.4%
AD_HYBRID	54.3%
AD_REC	54.2%
CHREV	32.7%
CF	25.4%
RF	15.2%

The best-performing baseline RF is the lowest in reasonable recommendation

AD_FREQ strikes the best balance between sticking to the ground truth and reasonable recommendations

AD_FREQ strikes the best balance between sticking to the ground truth and reasonable recommendations

Recommender	Percentage of Reasonable recommendations
KUREC	63.4%
ER	60.9%
AD_FREQ	59.4%
AD_HYBRID	54.3%
AD_REC	54.2%
CHREV	32.7%
CF	25.4%
RF	15.2%

The best-performing baseline RF is the lowest in reasonable recommendation

Summary of RQ3

KUREC is the recommender with the highest percentage of reasonable recommendations. Yet, **AD_FREQ** strikes the best balance between sticking to the ground truth and issuing reasonable recommendations when those deviate from that ground truth

We address three research questions (RQs)

RQ1: How accurately can KUREC recommend code reviewers in pull requests?

RQ2: Can KUREC be made more accurate by combining it with existing recommenders?

RQ3: How reasonable are those recommendations of KUREC that are not matched with ground truth data?

We address three research questions (RQs)

RQ1: How accurately can KUREC recommend code reviewers in pull requests?

RQ2: Can KUREC be made more accurate by combining it with existing recommenders?

RQ3: How reasonable are those recommendations of KUREC that are not matched with ground truth data?

Summary of RQ1

KUREC outperforms the remaining three baselines and has a more stable performance compared to RF, which is a desired property in practice

We address three research questions (RQs)

RQ1: How accurately can KUREC recommend code reviewers in pull requests?

RQ2: Can KUREC be made more accurate by combining it with existing recommenders?

RQ3: How reasonable are those recommendations of KUREC that are not matched with ground truth data?

Summary of RQ1

KUREC outperforms the remaining three baselines and has a more stable performance compared to RF, which is a desired property in practice

Summary of RQ2

Combining the KU-based recommender (KUREC) with the baselines in a straight-forward manner results in better-performing recommenders

We address three research questions (RQs)

RQ1: How accurately can KUREC recommend code reviewers in pull requests?

RQ2: Can KUREC be made more accurate by combining it with existing recommenders?

RQ3: How reasonable are those recommendations of KUREC that are not matched with ground truth data?

Summary of RQ1

KUREC outperforms the remaining three baselines and has a more stable performance compared to RF, which is a desired property in practice

Summary of RQ2

Combining the KU-based recommender (KUREC) with the baselines in a straight-forward manner results in better-performing recommenders

Summary of RQ3

KUREC is the recommender with the highest percentage of reasonable recommendations. Yet, **AD_FREQ** strikes the best balance between sticking to the ground truth and issuing reasonable recommendations when those deviate from that ground truth

We address three research questions (RQs)

RQ1: How accurately can KUREC recommend code reviewers in pull requests?

RQ2: Can KUREC be made more accurate by combining it with existing recommenders?

RQ3: How reasonable are those recomm
are not matched with grou

Summary of RQ1

KUREC outperforms the remaining three baselines and has a more stable performance compared to RF, which is a desired property in practice

Summary of

Combining the KU-based recommender (KUREC) with the baselines in a straight-forward manner results in better-performing recommenders

ary of RQ3

mender with the highest percentage of reasonable recommendations. Yet, **AD_FREQ** strikes the best balance between sticking to the ground truth and issuing reasonable recommendations when those deviate from that ground truth

md.ahasanuzzaman@queensu.ca