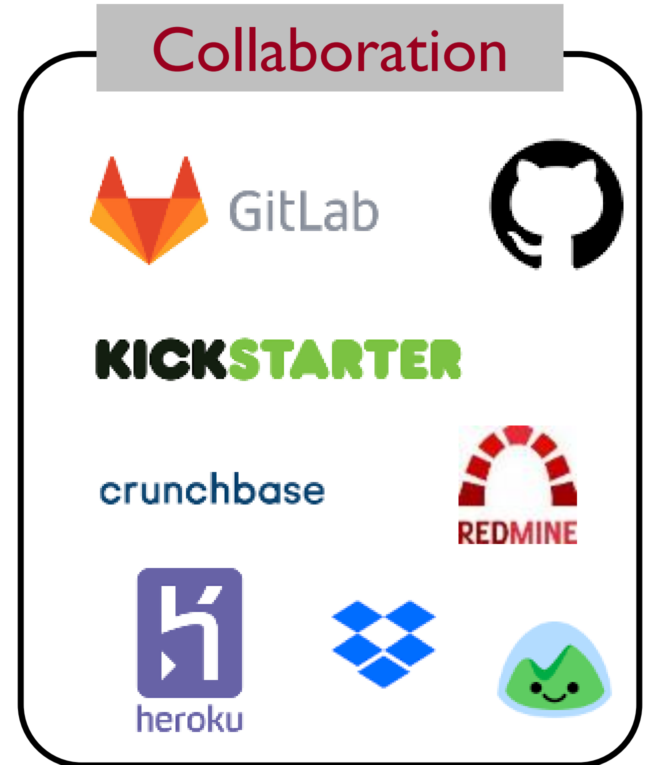
Tackling
Performance and Correctness Problems in Database-Backed Web Applications

Shan Lu (shanlu@uchicago.edu)

https://hyperloop-rails.github.io/

Web Applications



Performance is important

❝ Performance has directly impacted the company's bottom line.

Driving user growth with performance imp

How the BBC build

By Matthew Clark (Netmag) January 17, 20

BBC found they lost an additional **10%** of users for every **additional second** their site took to load.

COOK

"How is our website's speed affecting our business?"

COOK reduced average page load time by **850 ms** which increased conversions by **7%**

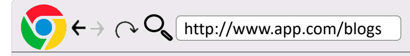
Walmart

Walmart found that for every **1 second** improvement in page load time, conversions increased by **2%,** which is **$200,000** increase in revenue

r revs up
rt size and sales for

For Mobify, every 100ms decrease increases 1.11% conversions and 38000$ annual revenue.

Performance issues happen all the time



This is Junwen’s defense.

Performance and correctness problem.

Junwen is graduating.

Hyperloop can improve perf of web app.

Panorama is view-centric.

PowerStation is a RubyMine plugin.

Traditional static webpage

blogs.html

```
<p> This is Junwen’s defense. </p>

<p> Performance and correctness problem </p>

<p> Junwen is graduating. </p>

<p> Hyperloop can improve perf of web app. </p>

<p> Panorama is view-centric </p>

<p> PowerStation is a RubyMine plugin</p>

…
```

Modern web app: dynamic content

User interface

Application Server

http://www.app.com/blogs

This is Junwen’s defense.

Performance and correctness problem.

Junwen is graduating.

Hyperloop can improve perf of web app.

Panorama is view-centric.

PowerStation is a RubyMine plugin.

blogs.html

```
<% @blogs.each do |blog|%>
  <p><%=blog.title %></p>
<% end %>
```

```
@blogs = read(‘blogs.json’)
```

```
blogs.json
{
"blog": {
   "title": "This is…"
 }
"blog": {
   "title": "Performance…"
 } …
}
```

http://www.app.com/blogs

This is Junwen's defense.

Performance and correctness problem.

Junwen is graduating.

Hyperloop can improve perf of web app.

Panorama is view-centric.

PowerStation is a RubyMine plugin.

Modern web app: big data

User interface

Application Server

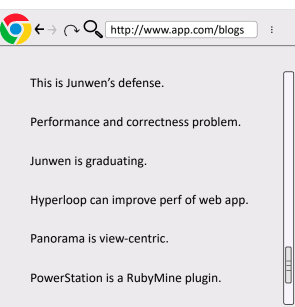
DB engine

```
<% @blogs.each do |blog|%>
  <p><%=blog.title %></p>
<% end %>
```

```
@blogs = read('blogs.json')
```

```
blogs.json
{
"blog": {
   "title": "This is."
 }
"blog": {
   "title": "Performance"
 } …
}
```

Modern web app: big data



```
<% @blogs.each do |blog|%>
  <p><%=blog.title %></p>
<% end %>
```

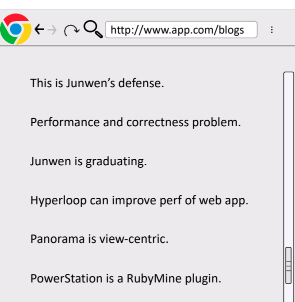
```
@blogs = read('blogs.json')
```

```
blogs.json
{
"blog": {
   "title": "This is."
 }
"blog": {
   "title": "Performance"
 } …
}
```

Table: blogs

id	title
1	This is Junwen's defense.
2	Performance and correctness
3	Junwen is graduating.
4	Hyperloop can improve …
…	…

Modern web app: big data



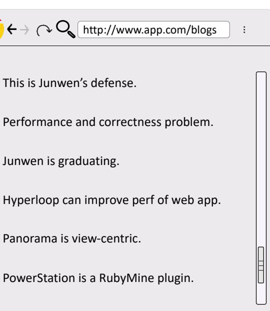
```
<% @blogs.each do |blog|%>
  <p><%=blog.title %></p>
<% end %>
```

```
@blogs = ???
```

```
select * from blogs
```

Table: blogs

id	title
1	This is Junwen's defense.
2	Performance and correctness
3	Junwen is graduating.
4	Hyperloop can improve …
…	…



Modern web app: big data

```
<% @blogs
   <p><%=l
<% end %
```

GitLab | Projects | Groups | Activity | Milestones | Snippets | This group | Search

GitLab.org > Issue Boards

11.3 Plan | Edit board | Add list

Backlog 5

- Remove ghost notification settings for groups and projects gitlab-org/gitlab-ce#44824 3 — Plan, Stretch, backend, database, notifications, technical debt
- Create RuboCop cop for url_for(params, ...) gitlab-org/gitlab-ce#47986 — Plan, backend, backstage, security, static analysis
- Consider updating all API endpoints that accept comma-separated strings to also accept arrays gitlab-org/gitlab-ce#48007 — Plan, Stretch, api, backend, enhancement
- Use serializer to render diff lines gitlab-org/gitlab-ce#48084 2 — Plan, backend, diff, technical debt
- Stored XSS on Issue details page gitlab-org/gitlab-ce#49422 — HackerOne, P2, Plan, S2, frontend, issues, markdown, security

performance 5

- Controller Projects::MergeRequestsController#ci_environments_status executes more than 100 SQL queries gitlab-org/gitlab-ce#43109 — P3, Plan, S3, backend, database, performance
- SQL timings in Projects::MergeRequestsController#show.json should be no more than 100ms per request gitlab-org/gitlab-ce#43394 — P3, Plan, S2, backend, database, performance
- Cache diff related data from blobs when loading "/diffs" gitlab-org/gitlab-ce#45693 — Gitaly, Plan, Stretch, backend, diff, performance
- SQL timings for Projects::IssuesController#show should be below 100 ms per request gitlab-org/gitlab-ce#48220 — Plan, S2, backend, database, issues, performance
- Limit the number of comments on a noteable gitlab-org/gitlab-ce#46676 — Deliverable, Plan, backend, issues, merge requests, performance

bug 7

- Postgres timeout when counting number of CI builds for usage ping gitlab-org/gitlab-ce#45938 — Plan, admin dashboard, backend, bug, reproduced on GitLab.com, usage ping
- Searching in a group does not search subgroups gitlab-org/gitlab-ce#47395 — Plan, backend, bug, search, subgroups
- Can't attach image to epic description and comment gitlab-org/gitlab-ee#7009 — Deliverable, Plan, backend, bug, due-22nd, epics, portfolio management
- Stack trace of error from uploading image for object storage spews into screen gitlab-org/gitlab-ee#6699 — Object Storage, Plan, backend, bug
- Better error handling for Elastic gitlab-org/gitlab-elasticsearch-indexer#19 — Deliverable, Plan, backend, bug, elasticsearch
- All Gitlab branches listed in Jira task in Development section gitlab-org/gitlab-ee#6752 — Deliverable, Plan, backend, bug, customer, jira
- GitLab does not add comment/link to Jira on mention in commit message

Deliverable 22

- Account for issues created in the middle of a milestone in burndown chart gitlab-org/gitlab-ee#6174 3 — Deliverable, Plan, backend, direction, due-22nd, milestones
- Order issues / merge requests / epics lists in both directions gitlab-org/gitlab-ce#39849 — Accepting Merge Requests, Deliverable, Plan, UX ready, backend, due-22nd, frontend, issues, merge requests
- Merge request list row design gitlab-org/gitlab-ce#47010 — Deliverable, Plan, UX ready, direction, due-22nd, frontend, merge requests
- Sort by start date and end date in the roadmap view and epics list view gitlab-org/gitlab-ee#6494 — Deliverable, Plan, UX, backend, due-22nd, epics, frontend, portfolio management, roadmaps
- Quick action to add/remove issue to epic from issue gitlab-org/gitlab-ee#6959 — Deliverable, Plan, UX ready, backend, direction, due-22nd, epics, frontend, issues, portfolio management, quick actions
- Can't attach image to epic description and

DB engine

```
* from blogs
```

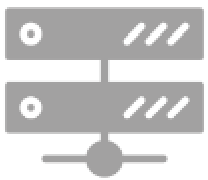
Table: blogs

wen's defense.
ce and correctness
graduating.
can improve ...
... ...

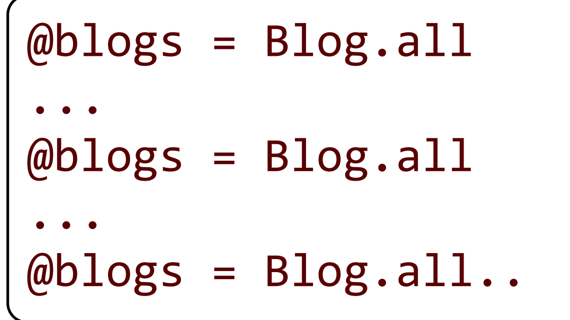
Performance challenges (1)

Treating ORM APIs as a black box

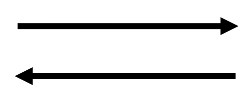
What queries?



Inefficient application code

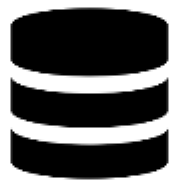


```
@blogs = Blog.all
...
@blogs = Blog.all
...
@blogs = Blog.all..
```



> 10X slow down!

Application semantics?

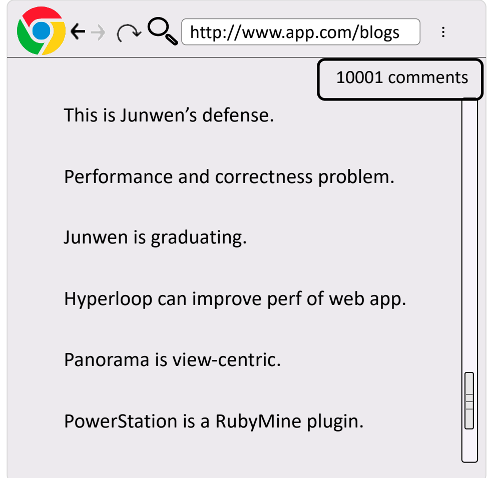


Inefficient database design

```
select * from blogs
...
select * from blogs
...
select * from blogs
```

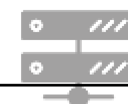
Lack of App-DB cross-stack optimization

Performance challenges (2)



Generating the comment count tag costs 1.2s on a 20k-record database!

```
<p> <%= @count %> comments</p>
```



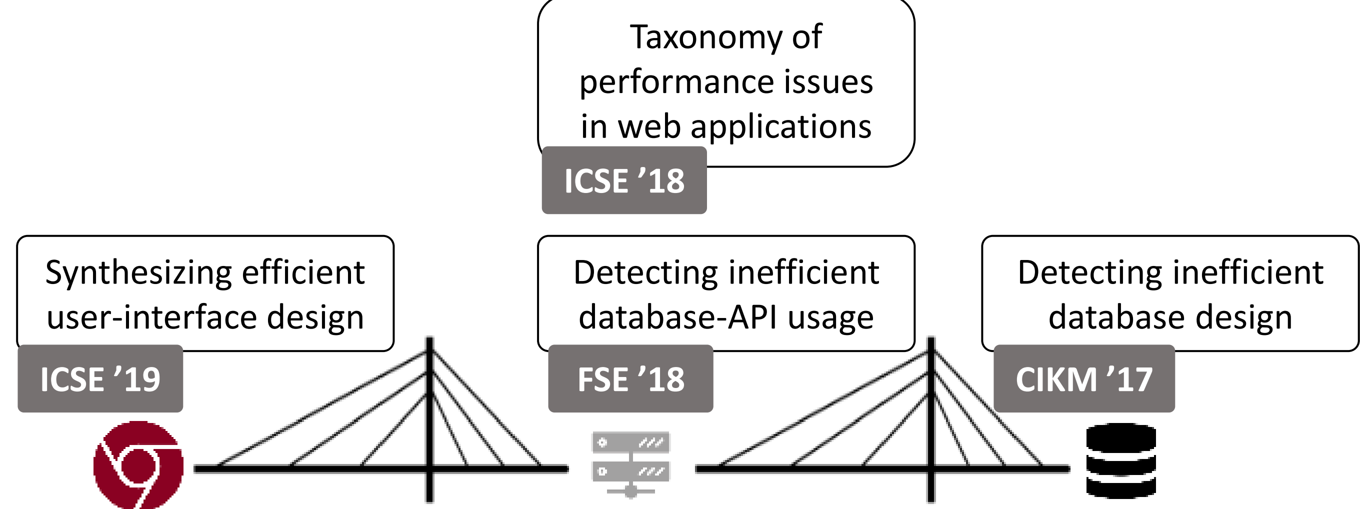
```
@count=Comment.join(…).where(…)
```

```
select count(*) from comments
join blogs on blogs.comment_id
= comments.id where is_deleted
= false
```

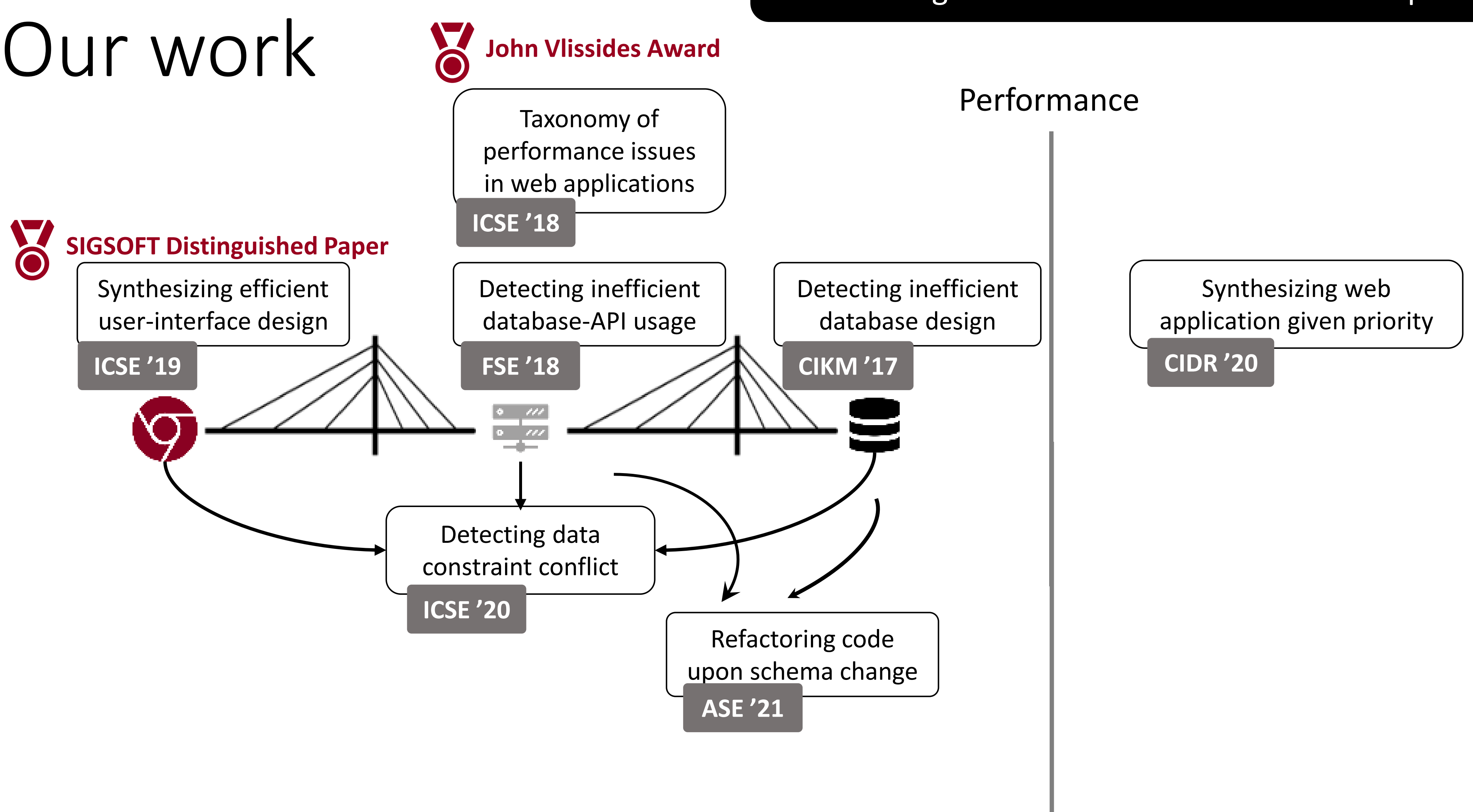
Lack of DB-aware user-interface optimization

Improving performance and correctness of web applications using **cross-stack** and **user-interface** optimization

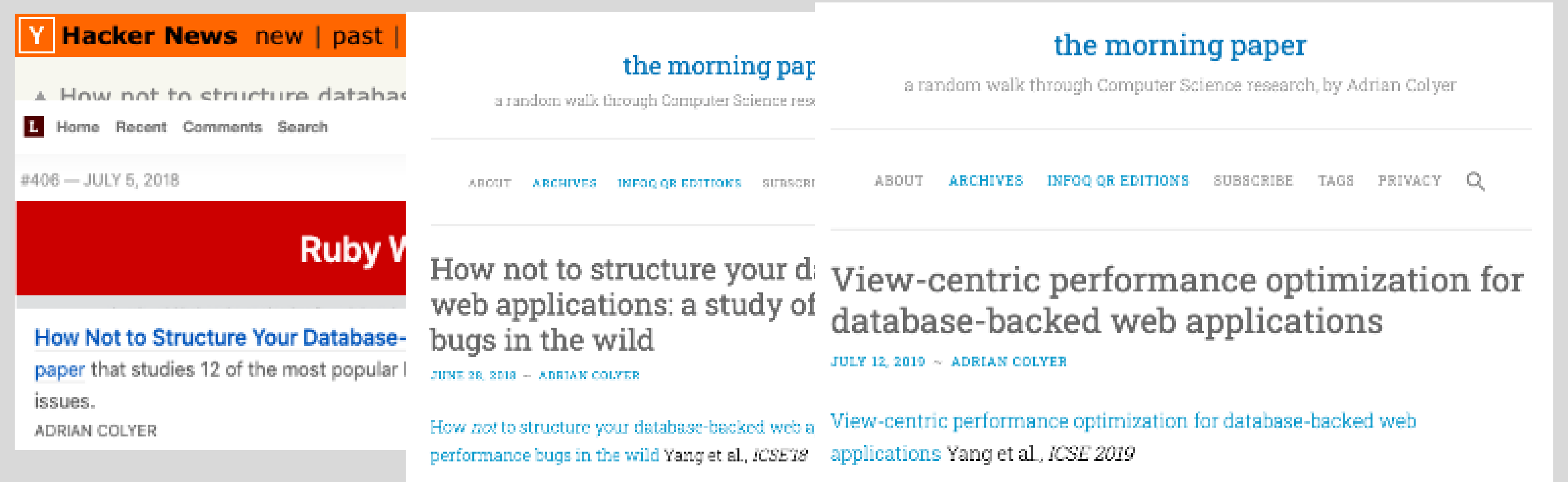
Our work



Our work

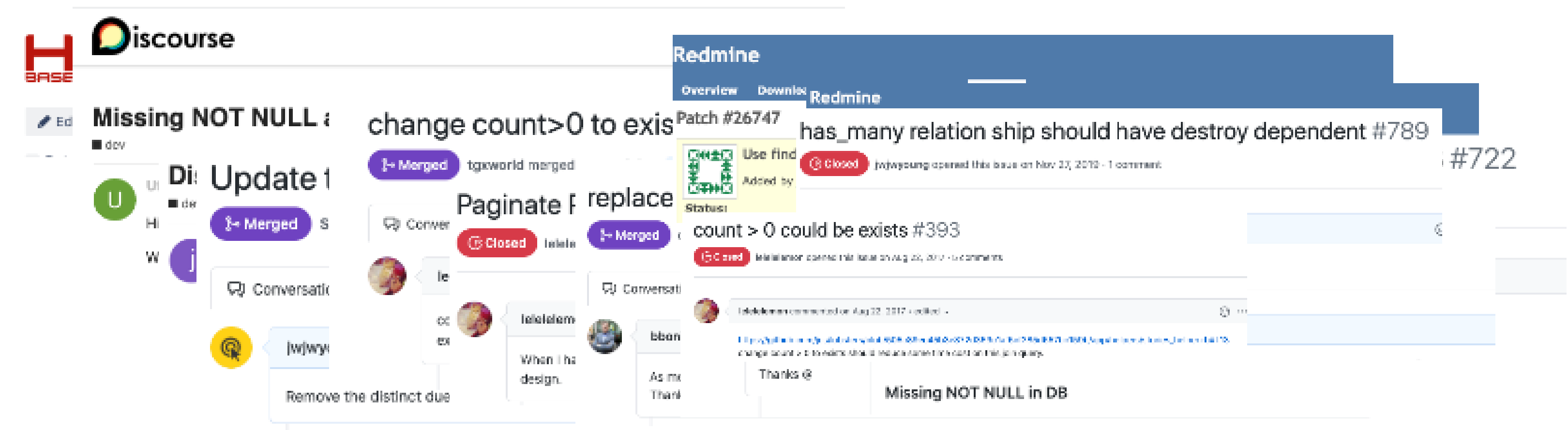


Raised attention in open-source community, HackerNews, RubyWeekly, morning paper

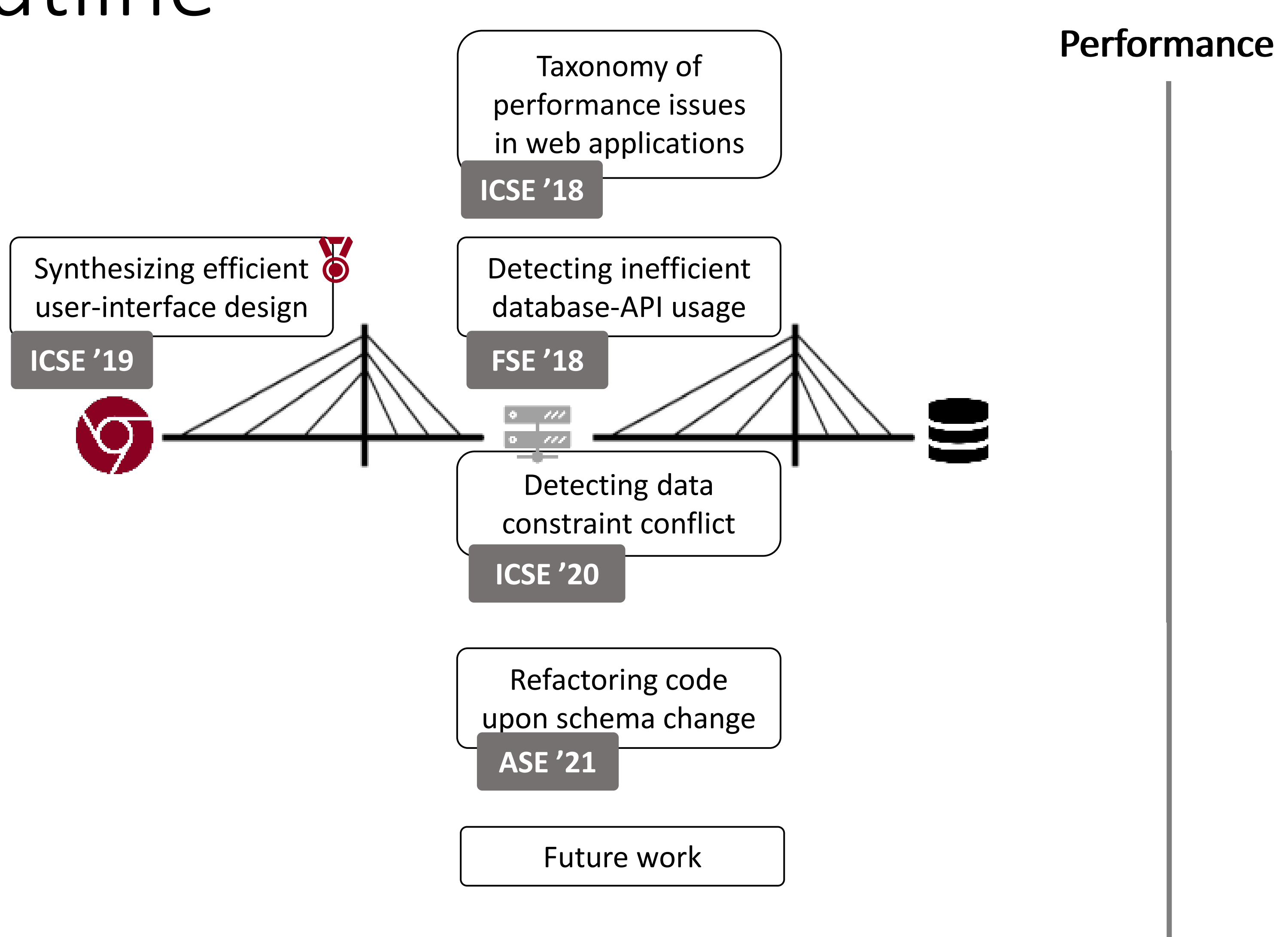


IMPACT

Detected thousands of unknown bugs from Discourse, Redmine, …



Outline

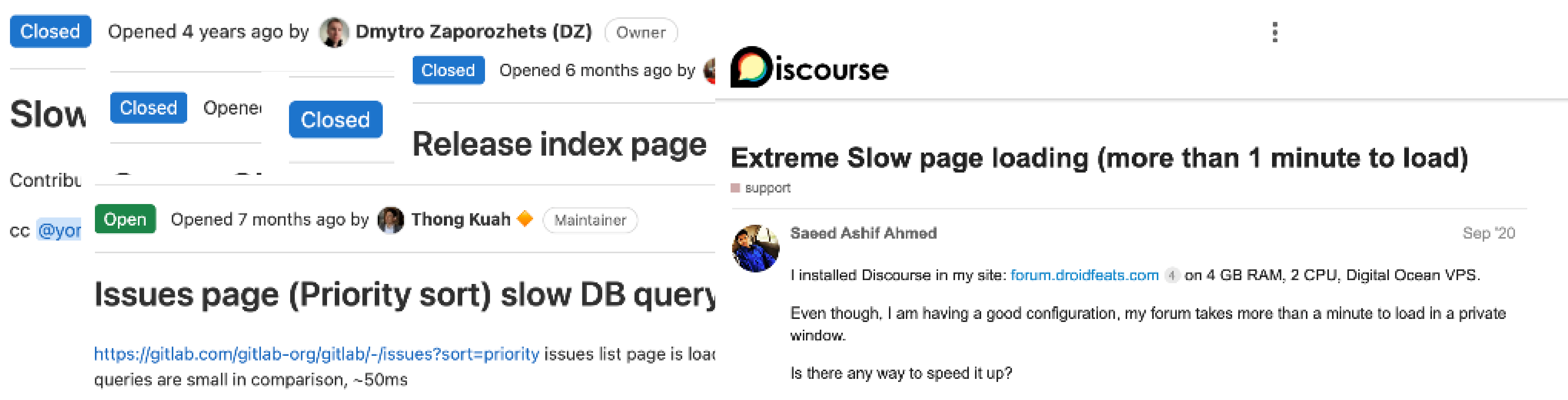


Performance

Correctness

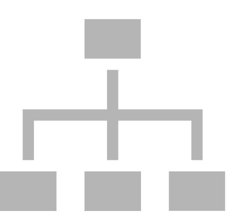
Understanding performance problems

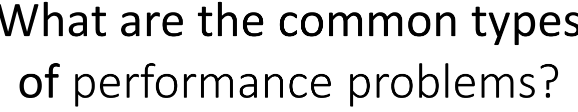
- **Why?** *many complaints and yet no comprehensive studies*



How not to structure your database-backed web applications: a study of performance bugs in the wild. ICSE '18
Yang Junwen, Cong Yan, Pranav Subramaniam, Shan Lu, and Alvin Cheung.

Goals of our study





What are the common types of performance problems?

How severe are **the performance** problems?

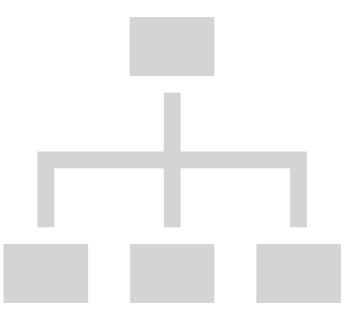
How complicated are the fixes?

Methodology

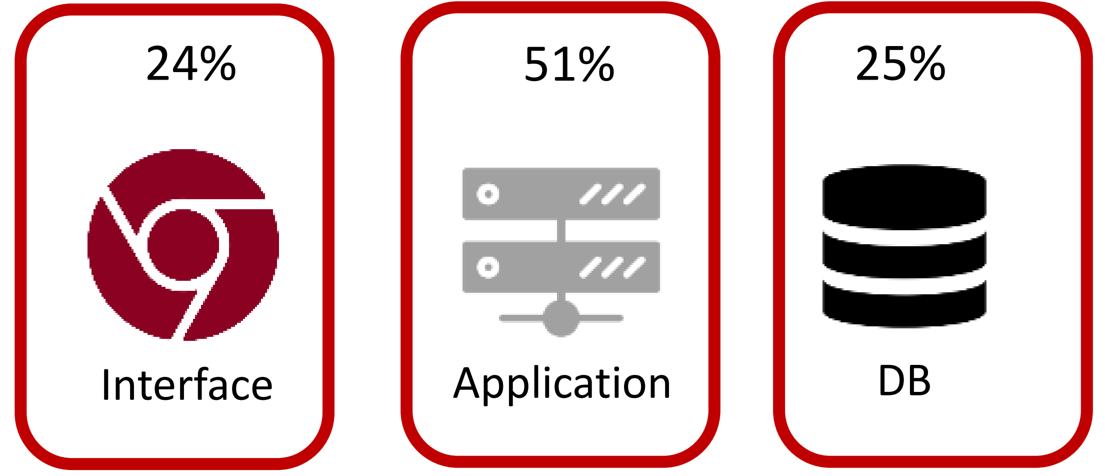
- 12 most popular open-source Rails applications
- 204 real-world performance problems
  - 140 known problems in old software versions
  - 64 un-fixed problems in latest software versions
    - Discovered by our profiling

Category	Name	Stars
Forum	Discourse	22k
	Lobsters	2.4k
Collaboration	Gitlab	49k
	Redmine	3.6k
E-commerce	Spree	17k
	Ror-ecommerce	1.7k
Task-management	Tracks	3.5k
	Fulcrum	697
Social Network	Diaspora	18k
	Onebody	1.2k
Map	Openstreetmap	8k
	Falling-Fruit	1.1k

What are the common types of inefficiency?

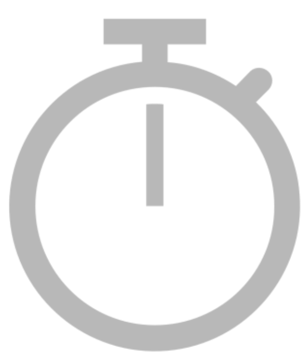


Anti-patterns across 3 layers

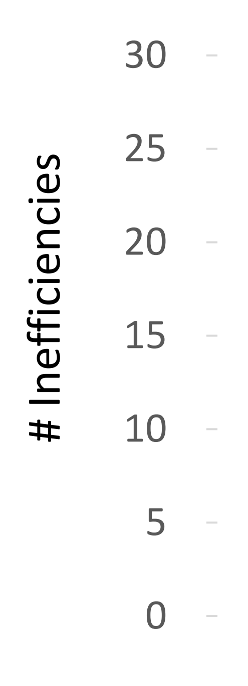


They will be introduced together with how we tackle them

How severe are these inefficiencies?



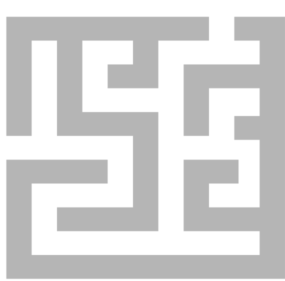
They are very costly!



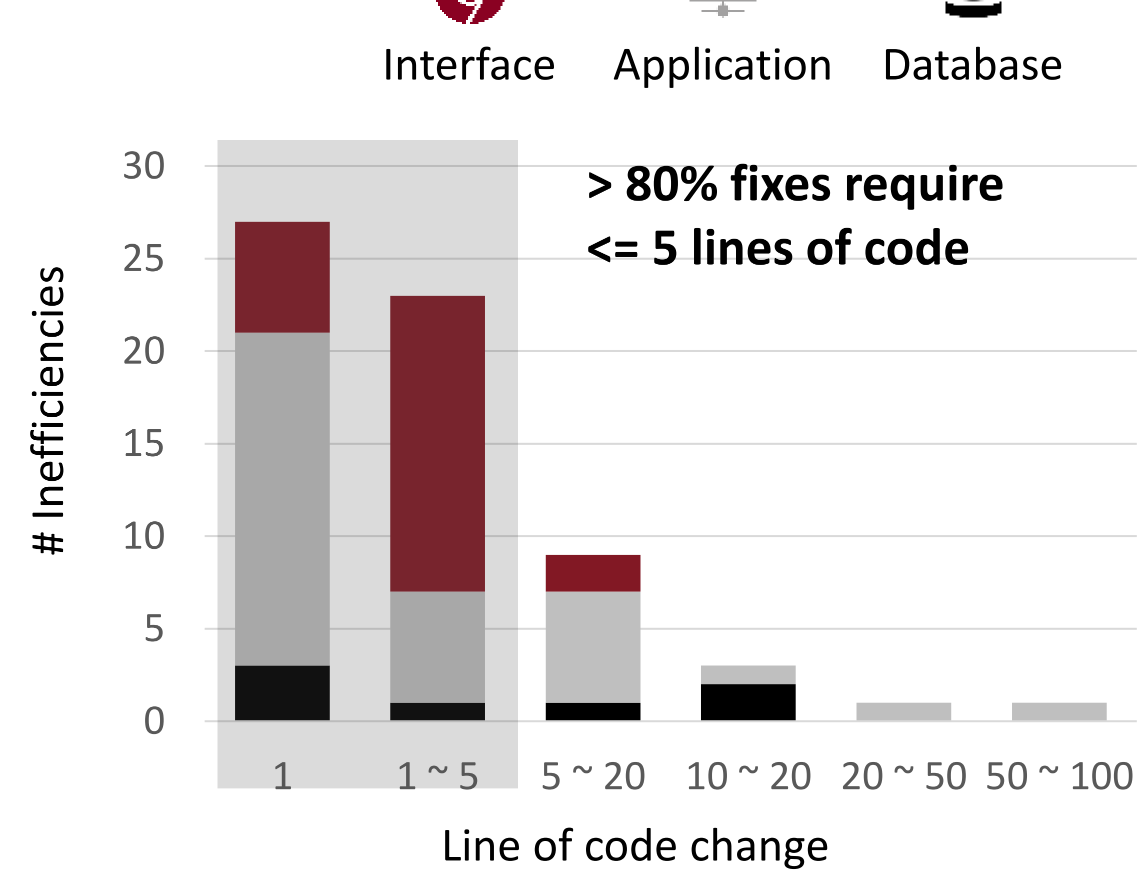
> 60% cause

Slowdowns

How complicated are the fixes?



Many patches are small!



- First comprehensive study of web apps' performance problems
- Motivation and guidance for follow-up research

IMPACT

- Well received by real-world web developers

This is cool. Are there more papers as practically useful for a (web) developer as this one? [1]

I think the article alludes to much more important problems like querying in a loop, querying the same information again just because it's not in function/object scope [1]

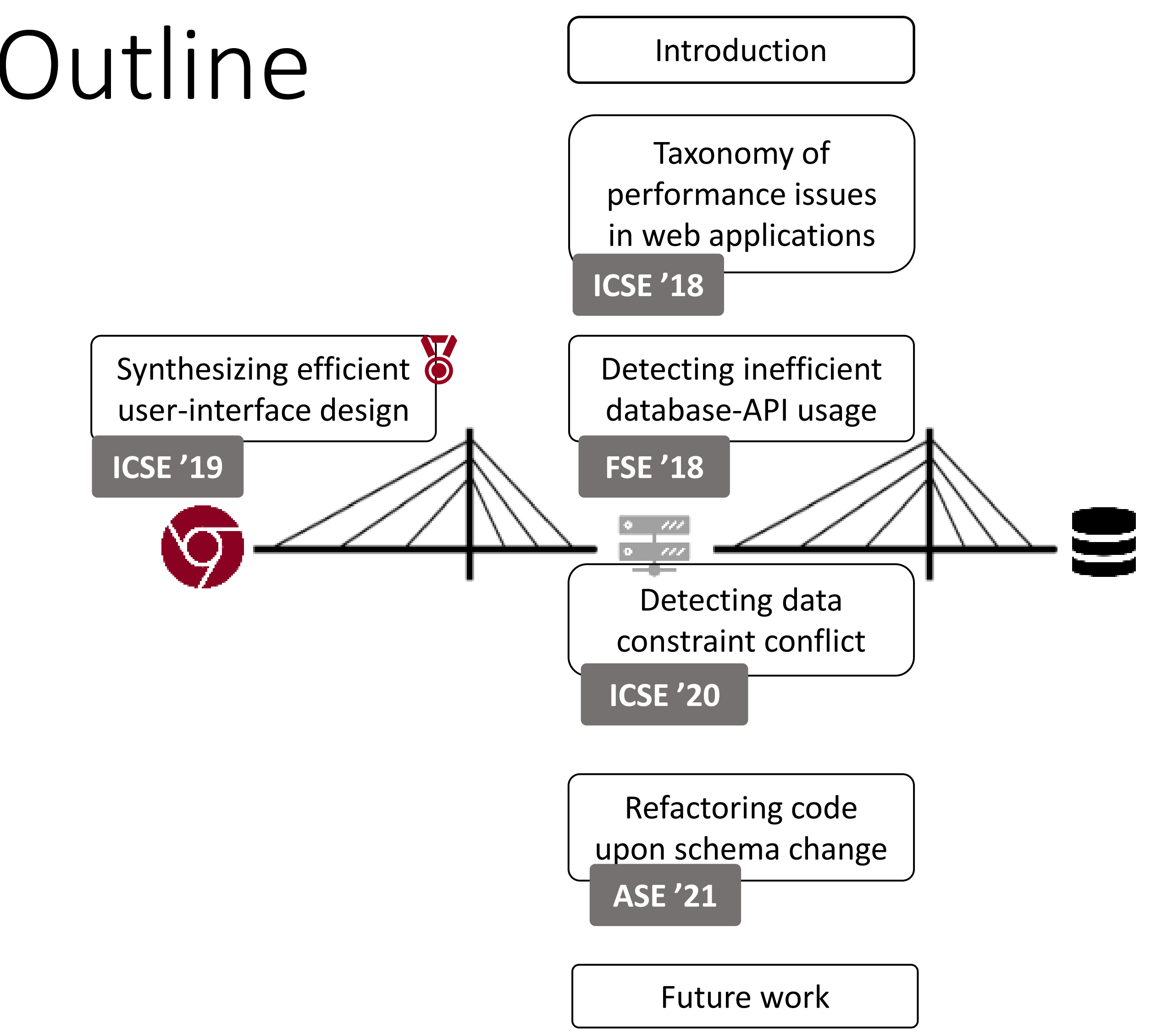
The article helped me to understand the com
ORM such as ActiveRecord. I've found this ve
the experiences that I shared are helpful to sc
tooling will be built to help us, developers, avc

Most scientific papers are unlikely to change your day-to-day approach as a Rails web developer. How not to structure your database-backed web applications: a study of performance bugs in the wild Yang et al., ICSE'18 is the exception to that rule. [3]

1. https://news.ycombinator.com/item?id=17414383
2. https://www.yoranbrondsema.com/post/reflection-on-how-not-to-structure-your-database-backed-web-applications/
3. https://scoutapm.com/blog/part-i-how-not-to-structure-your-database-backed-web-apps

Performance

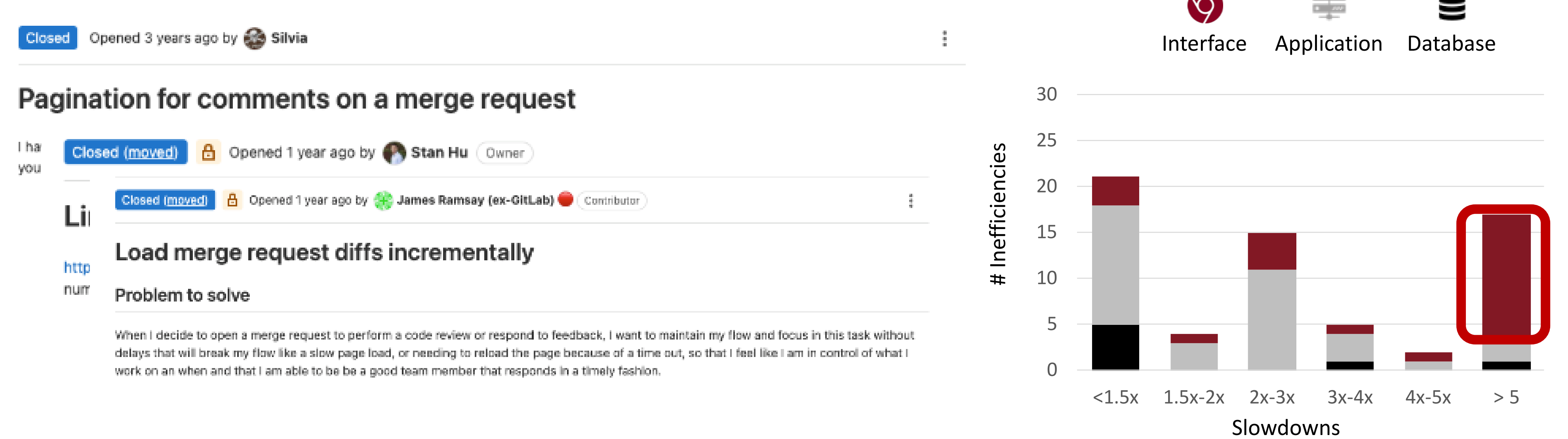
Outline



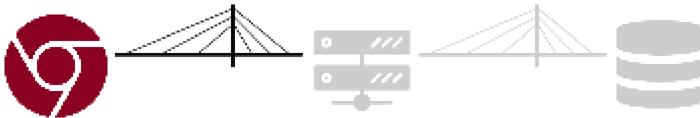
Correctness

Detecting and fixing performance-unfriendly interfaces

- **Why?** User-interface optimization brings large performance gains

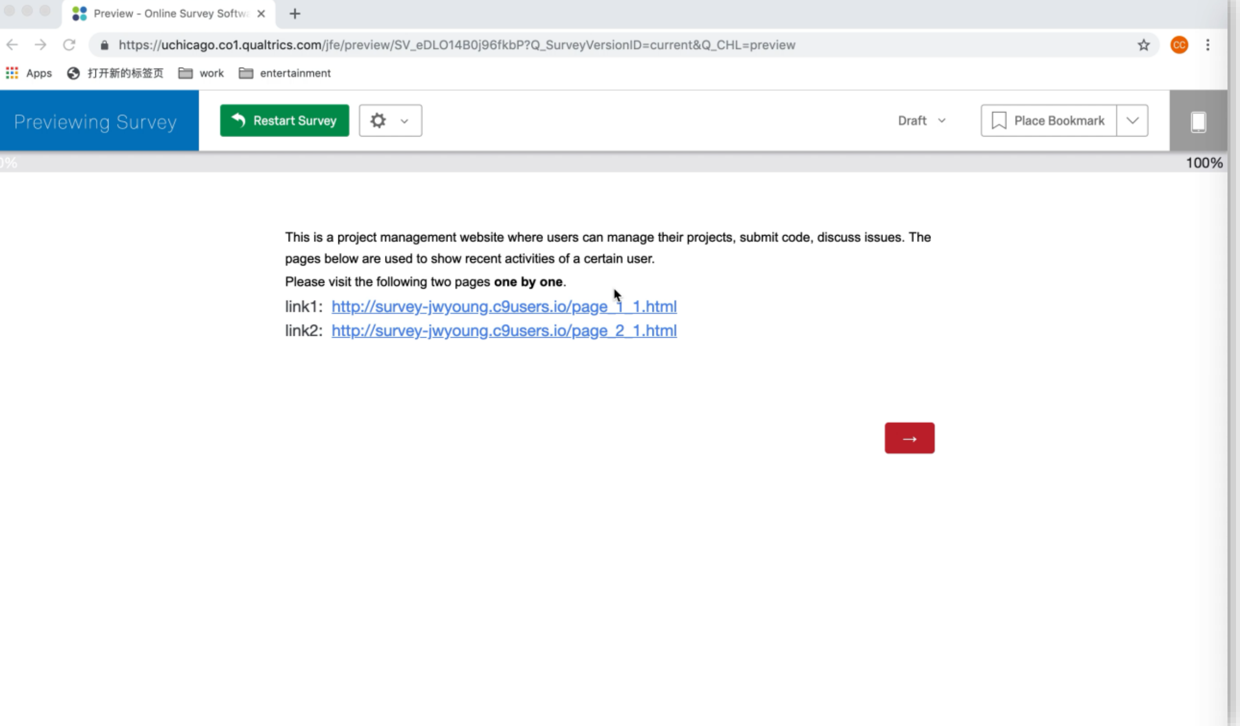


View-centric performance optimization for database-backed web applications. ICSE '19
Yang Junwen, Cong Yan, Chengcheng Wan, Shan Lu, and Alvin Cheung.

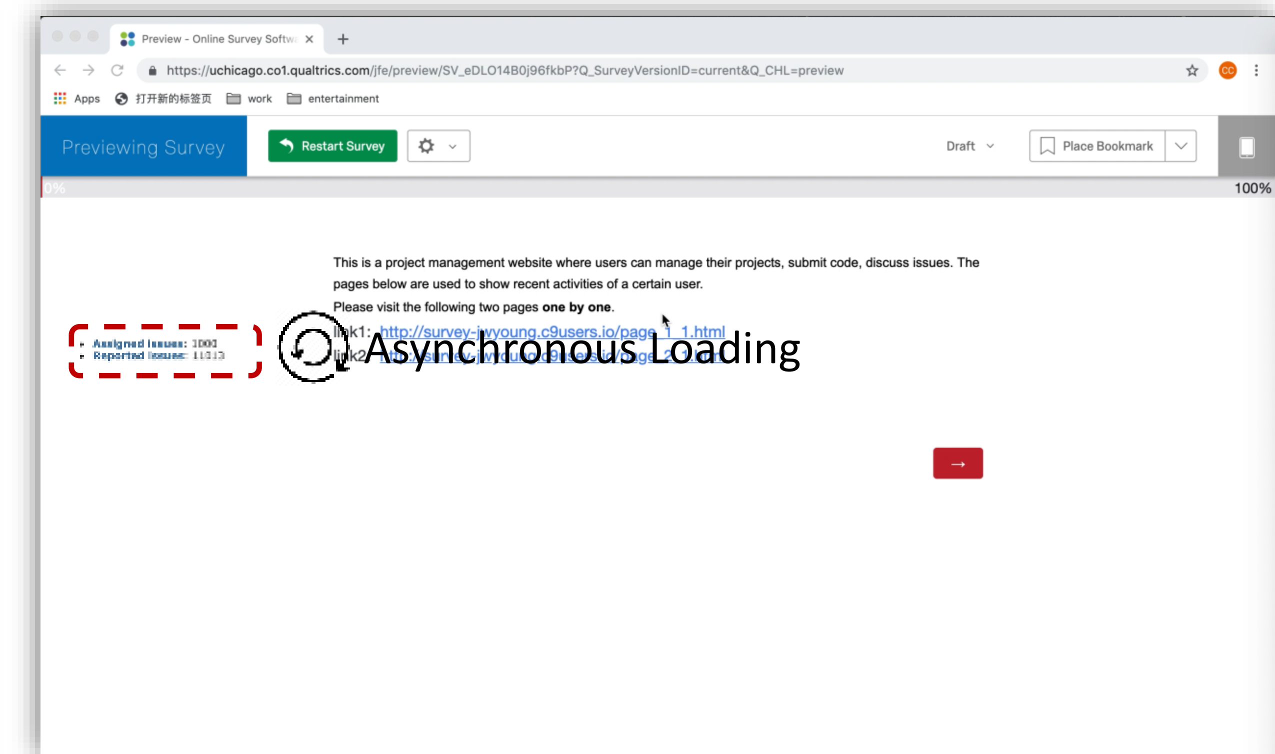


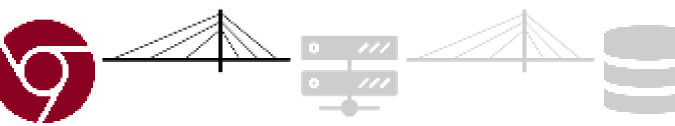
Examples

1



2

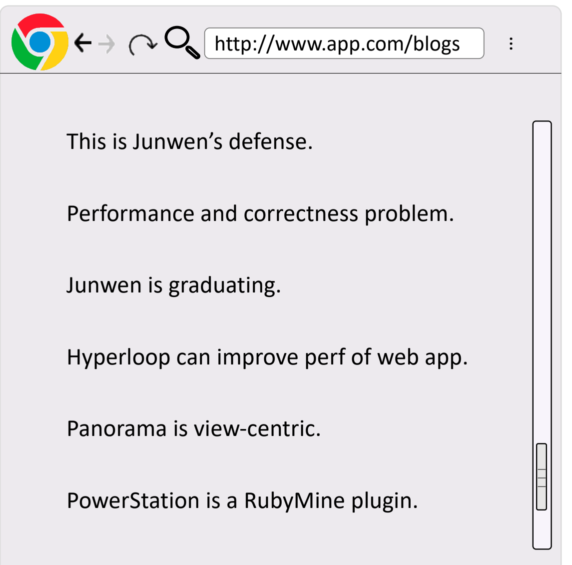


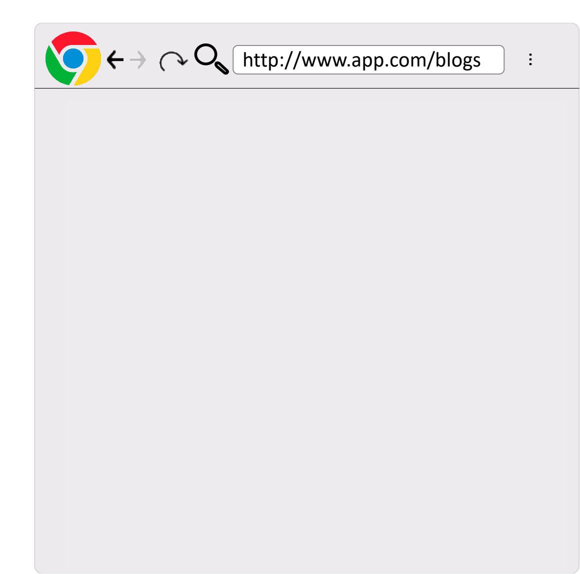


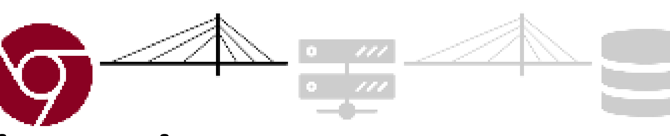
Can we automate interface optimization

1. Interface Usability vs. Performance

Out of the scope of traditional code optimization





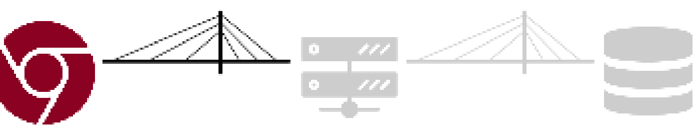


Can we automate interface optimization

1. Interface Usability vs. Performance

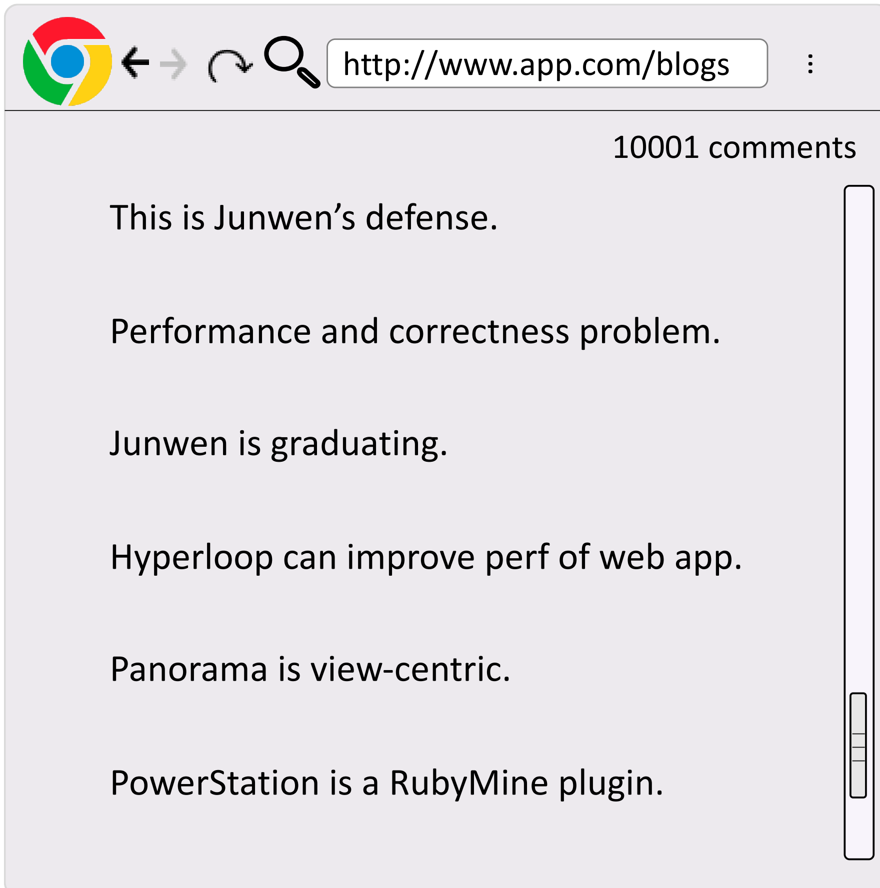
2. Code analysis & transformation across HTML, Ruby, SQL

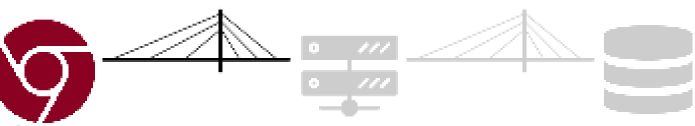
Extend HTML parser to fully understand Ruby code and SQL?



Solutions: a new interface design framework

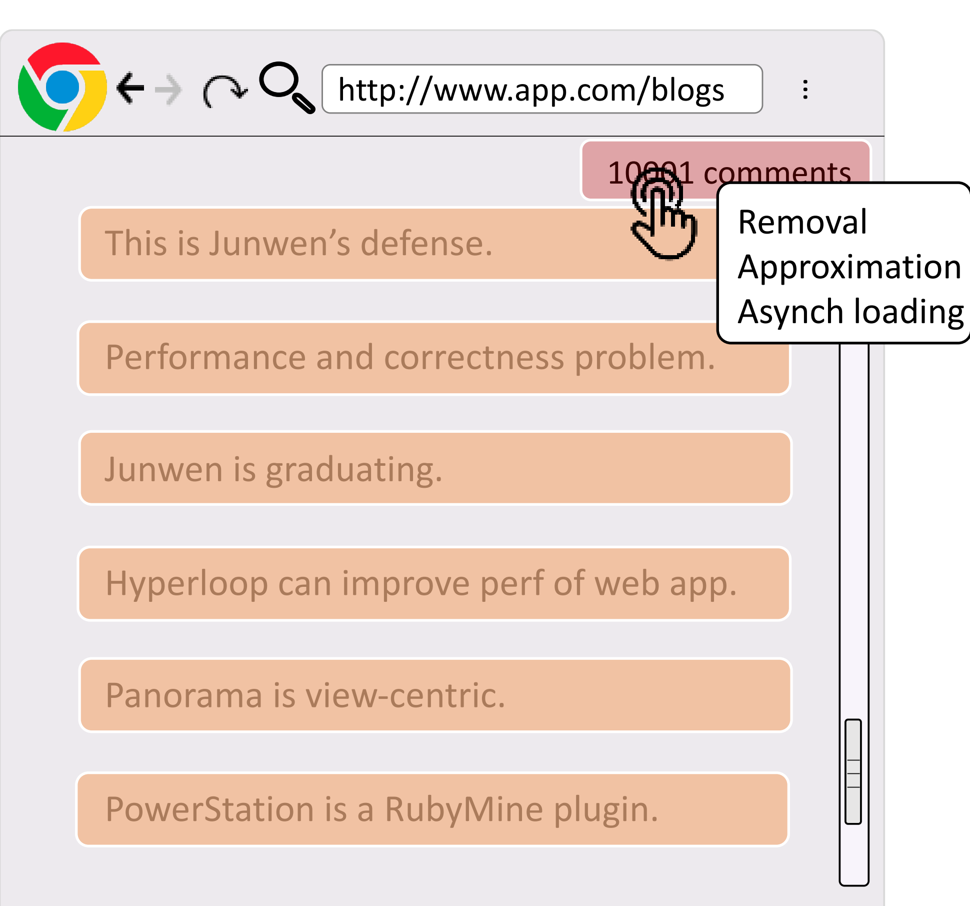
- Cost estimation & visualization

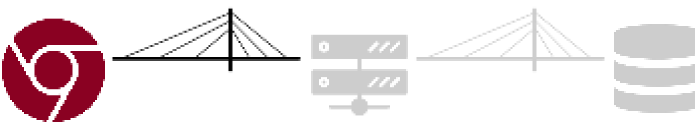




Solutions: a new interface design framework

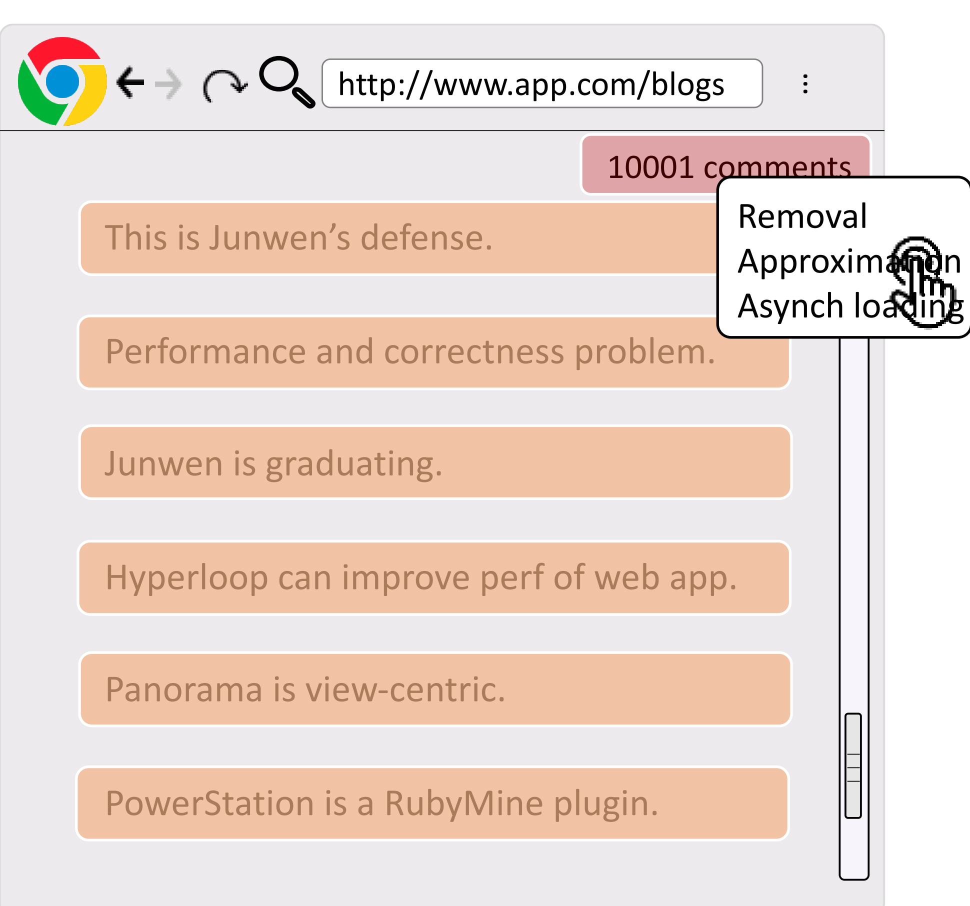
- Cost estimation & visualization
- Interface refactoring

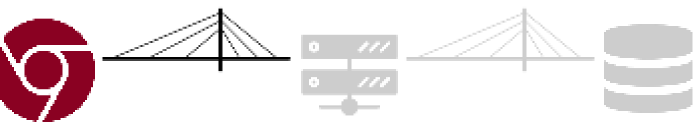




Solutions: a new interface design framework

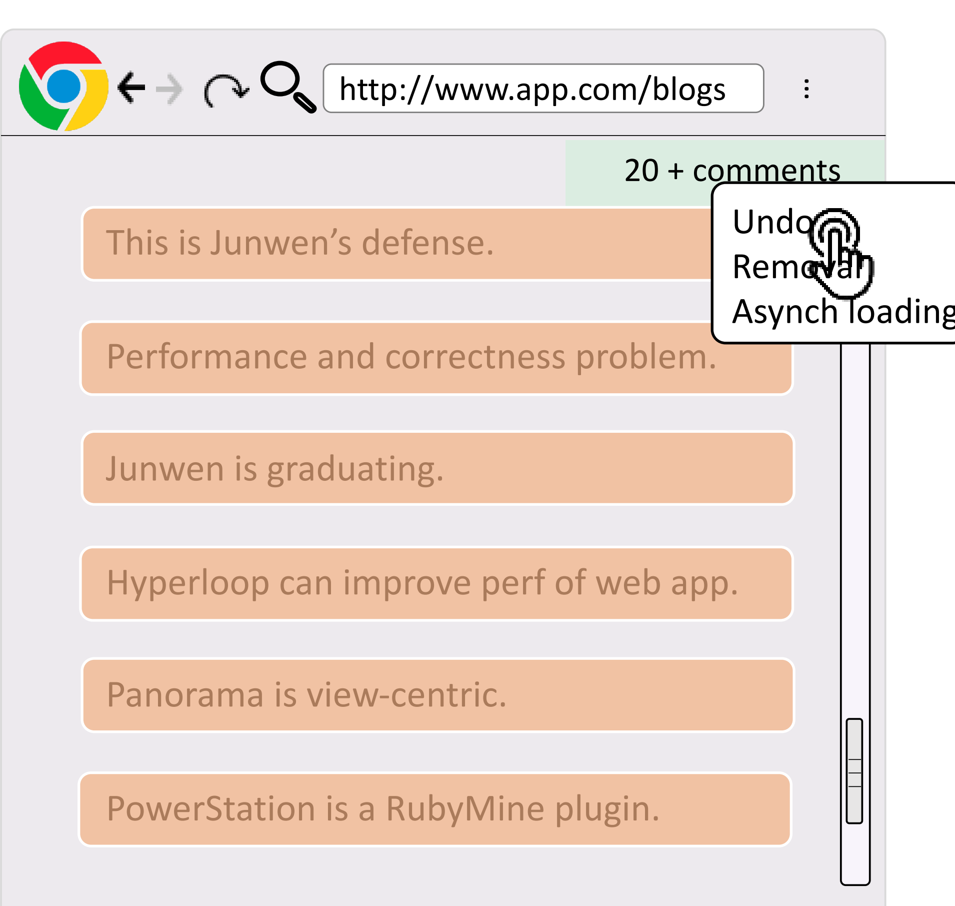
- Cost estimation & visualization
- Interface refactoring

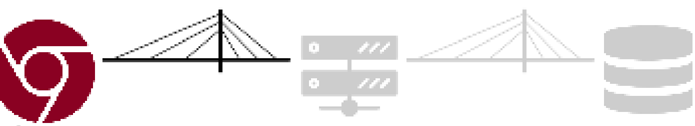




Solutions: a new interface design framework

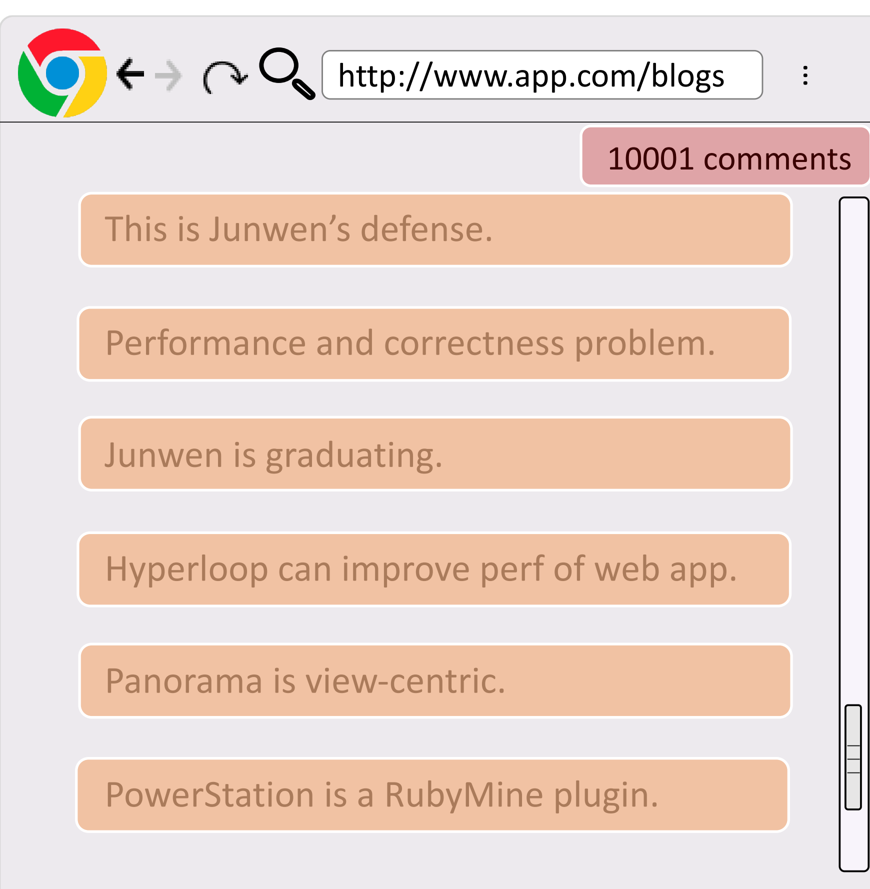
- Cost estimation & visualization
- Interface refactoring

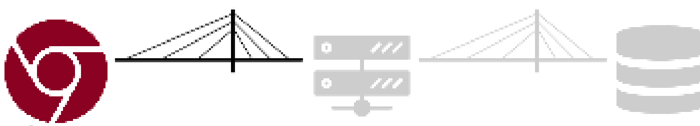




Solutions: a new interface design framework

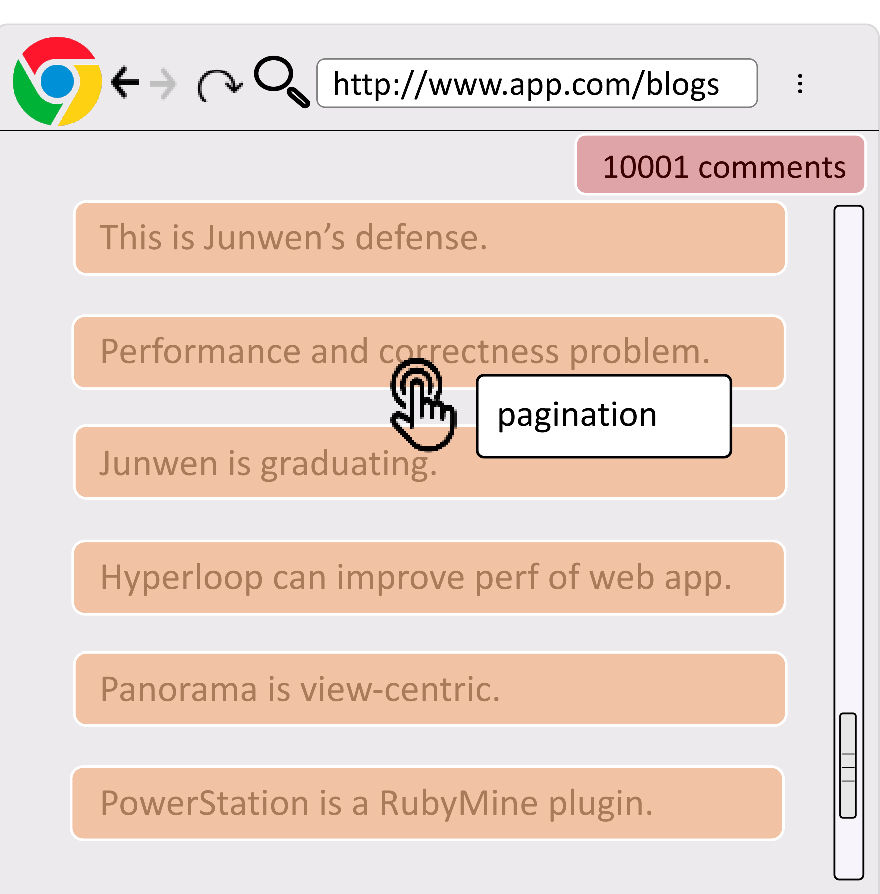
- Cost estimation & visualization
- Interface refactoring

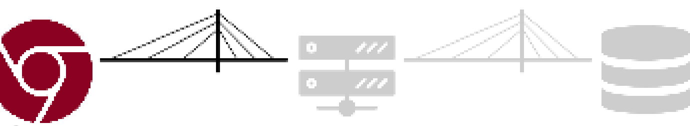




Solutions: a new interface design framework

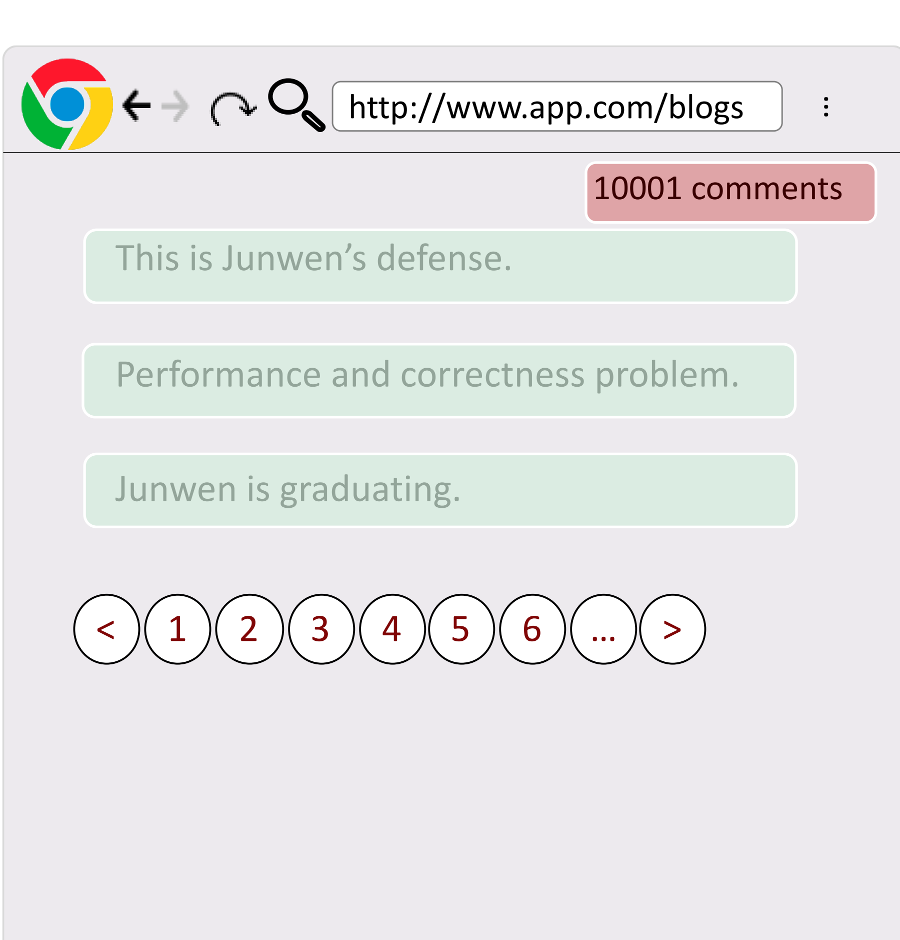
- Cost estimation & visualization
- Interface refactoring



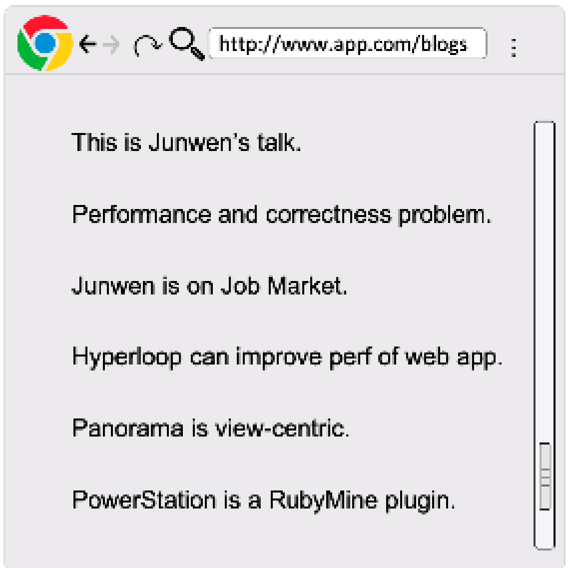


Solutions: a new interface design framework

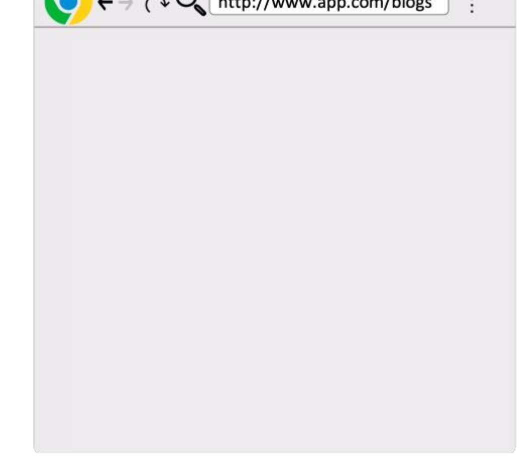
- Cost estimation & visualization
- Interface refactoring
  - WHAT
  - WHEN
  - HOW



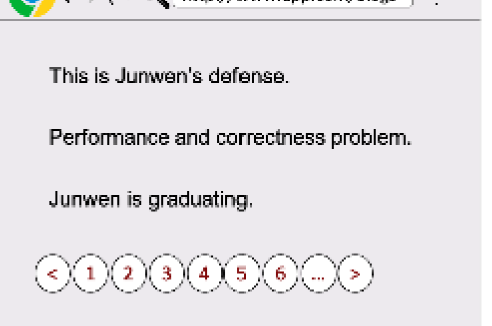
What are meaningful refactorings?



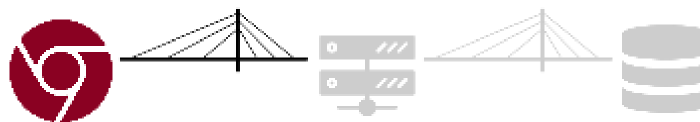
Not an optimization



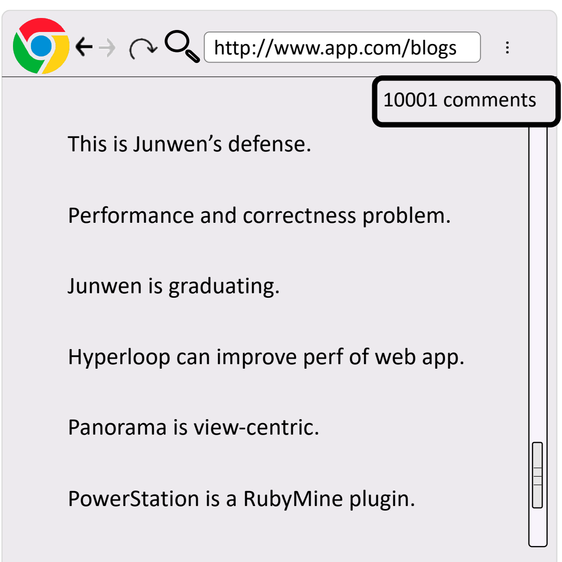
Optimization



Pagination

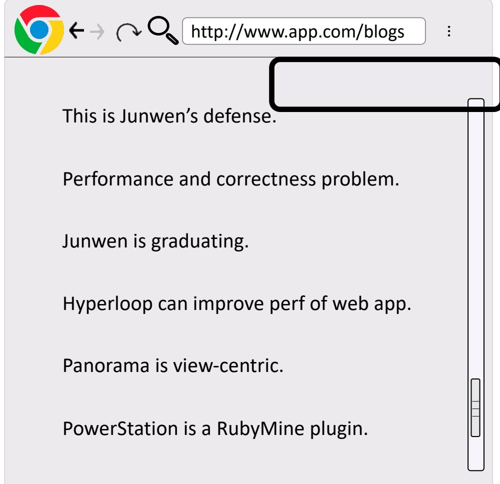


User-interface refactoring taxonomy

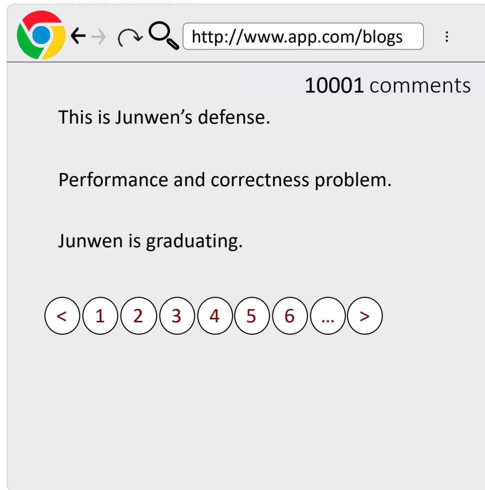


Displaying different format

Asynchronous Loading

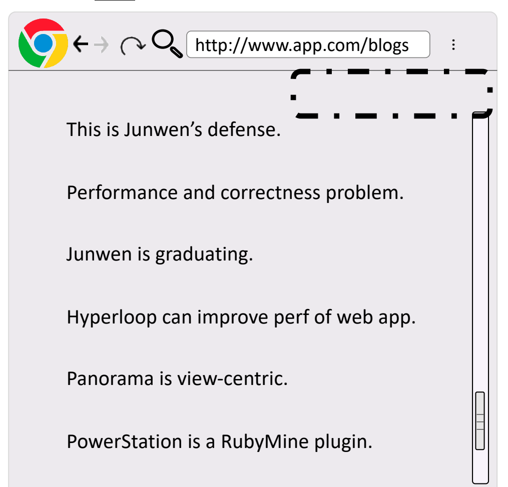


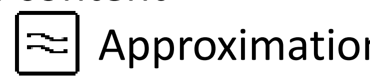
Paginating



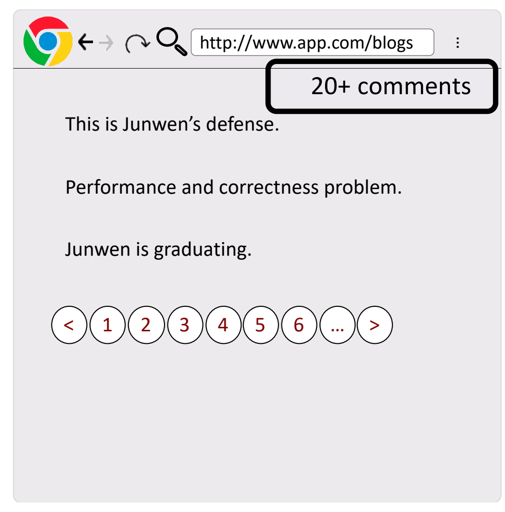
Displaying simplified content

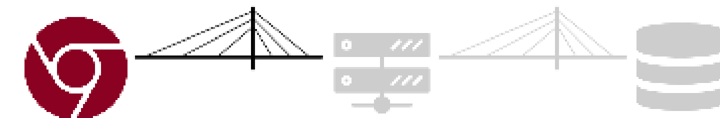
Tag Removal



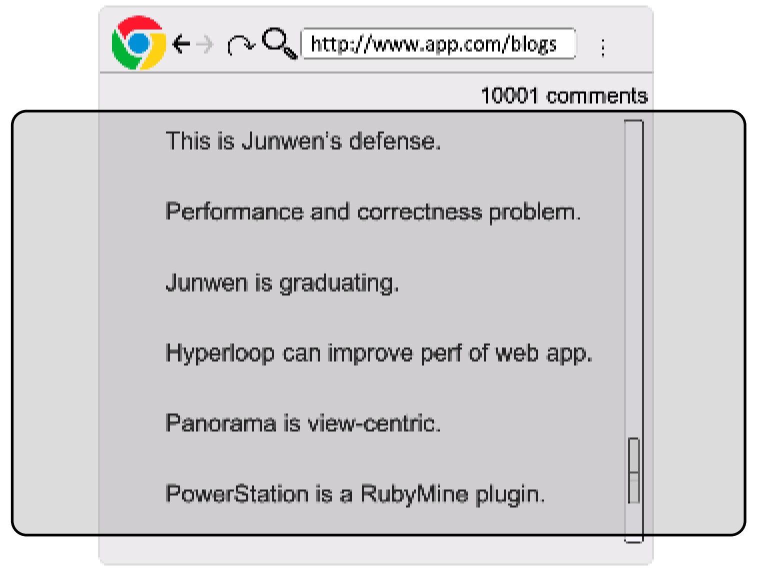


Approximation



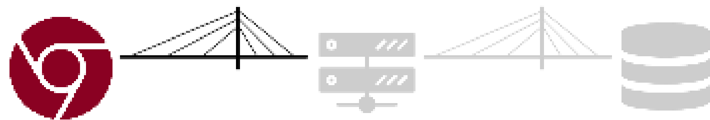


When to conduct refactoring?



Paginating: a long list of items on webpage

How to map interface design strategy to program analysis routines?



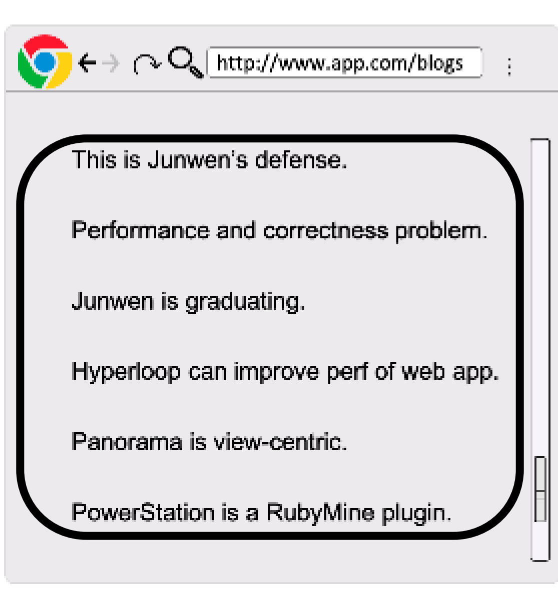
When to conduct pagination

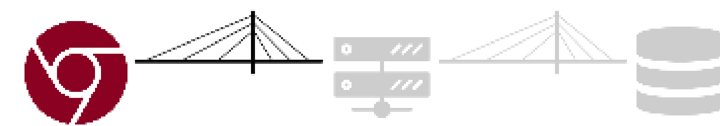
blogs_controller.rb

```
def index


  # Get a user’s blogs
  @blogs =  Blog.where(uid:?)
  render `index.html.erb`
end
```

index.html.erb

```
# render the blogs

<% @blogs.each do |blog| %>
<span id=‘blog’><%= blog.title %></span>
<% end %>
```





When to conduct pagination?

a long list of items on webpage

long	Scaling query results?

a list of items on webpage	A loop of html component?

blogs_controller.rb

```
def index

  # Get a user's blogs
  @blogs =  Blog.where(uid:?)
  render `index.html.erb`
end
```

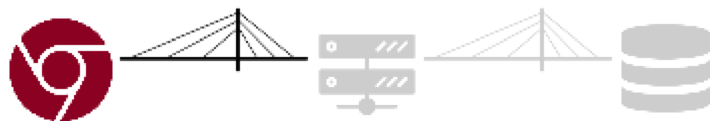
select * from blogs where uid = ?

HOW2. Is this statement mapped to a scaling query?

index.html.erb

```
# render the blogs

<% @blogs.each do |blog| %>
<span id='blog'><%= blog.title %></span>
<% end %>
```

HOW1. Which application code is used to generate which HTML tag?



When to conduct pagination?

http://www.app.com/blogs

high

low

1000 com

Removal
Approximation
Asynch loading

This is Junwen's defense.

Performance and correctness problem.

Junwen is graduating.

Hyperloop can improve per

pagination

Panorama is view-centric.

PowerStation is a RubyMine plugin.

blogs_controller.rb

```
def index
  # Get the count of comments of the user
  @count =  Comment.join(:blogs).where(uid:?).count
  # Get a user's blogs
  @blogs =  Blog.where(uid:?)
  render `index.html.erb`
end
```

select count(*) from blogs join comments …

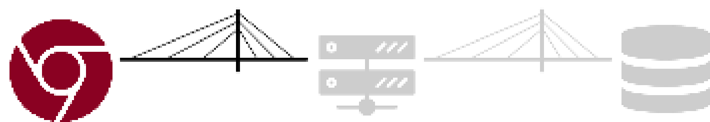
Not a scaling query

index.html.erb

```
# render the blogs
<p id='count'><%= @count %> comments</p>
<% @blogs.each do |blog| %>
  <span id='blog'><%= blog.title %></span>
<% end %>
```

Not a loop

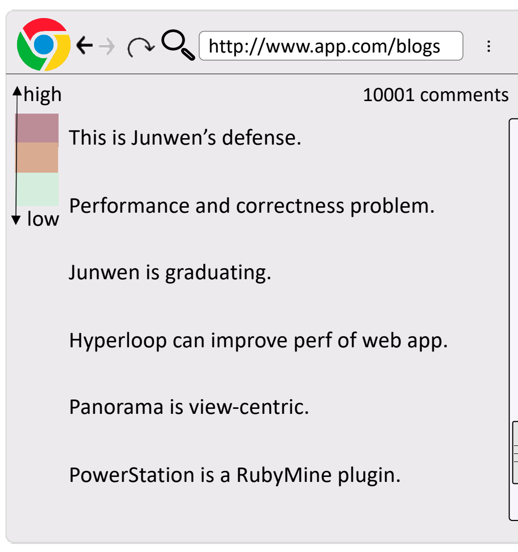
View-centric performance optimization, **ICSE '19**

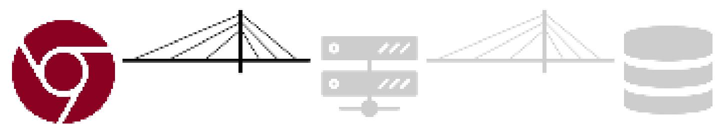


When to conduct asynch load?

Conceptually, every tag can be!

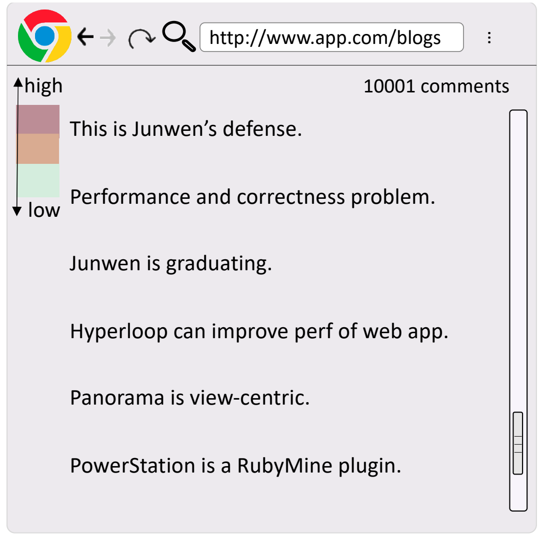
The tag should be time-consuming





When to conduct asynch load?

The tag should be time-consuming



- Estimating the cost of query nodes based on API call chain
- Propagating the cost to HTML nodes based on data dependency

O(N^1)

blogs_controller.rb

```
def index
  # Get the count of comments o
  @count =  Comment.join(:blogs)
  # Get a user's blogs
  @blogs =  Blog.where(uid:?)
  render `index.html.erb`
end
```

HOW1. Which application code is used to generate which HTML tag?

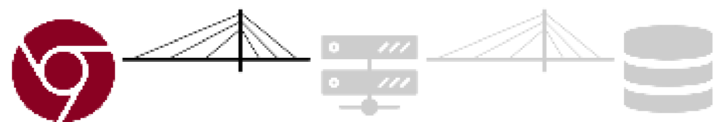
HOW3: What is the complexity of the application code?

index.html.erb

```
# render the blogs
<p id='count'><%= @count
<% @blogs.each do |blog| %>
  <span id='blog'><%= b                  >
<% end %>
```

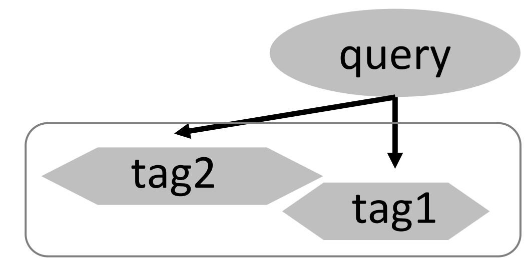
To be asynch loaded

Not to be asynch loaded

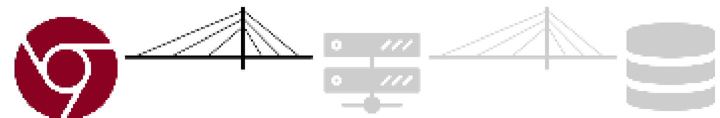


When to conduct asynch load?

The tag should be independent



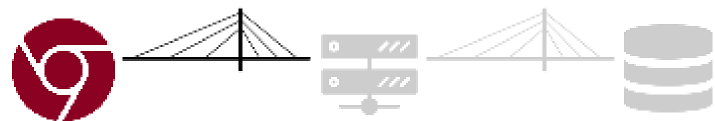
Asynch tag1 != Asynch query



How to conduct refactoring?

- HOW1: Which application code is used to generate which HTML tag?
- HOW2: Is this statement mapped to a scaling query?
- HOW3: What is the complexity of the application code?

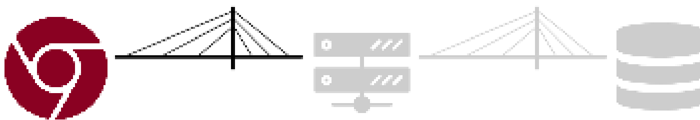
Extend Ruby compiler to understand selected HTML/ORM/SQL information



How to conduct refactoring?

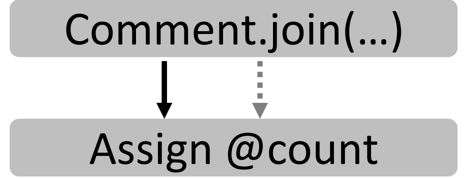
- HOW1: Which application code is used to generate which HTML tag?
- HOW2: Is this statement mapped to a scaling query?
- HOW3: What is the complexity of the application code?

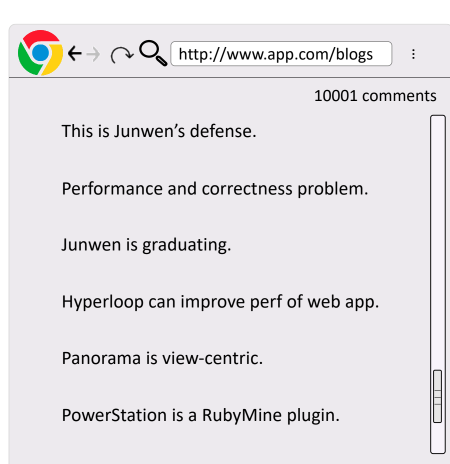
Extend Ruby compiler to understand selected HTML/ORM/SQL information

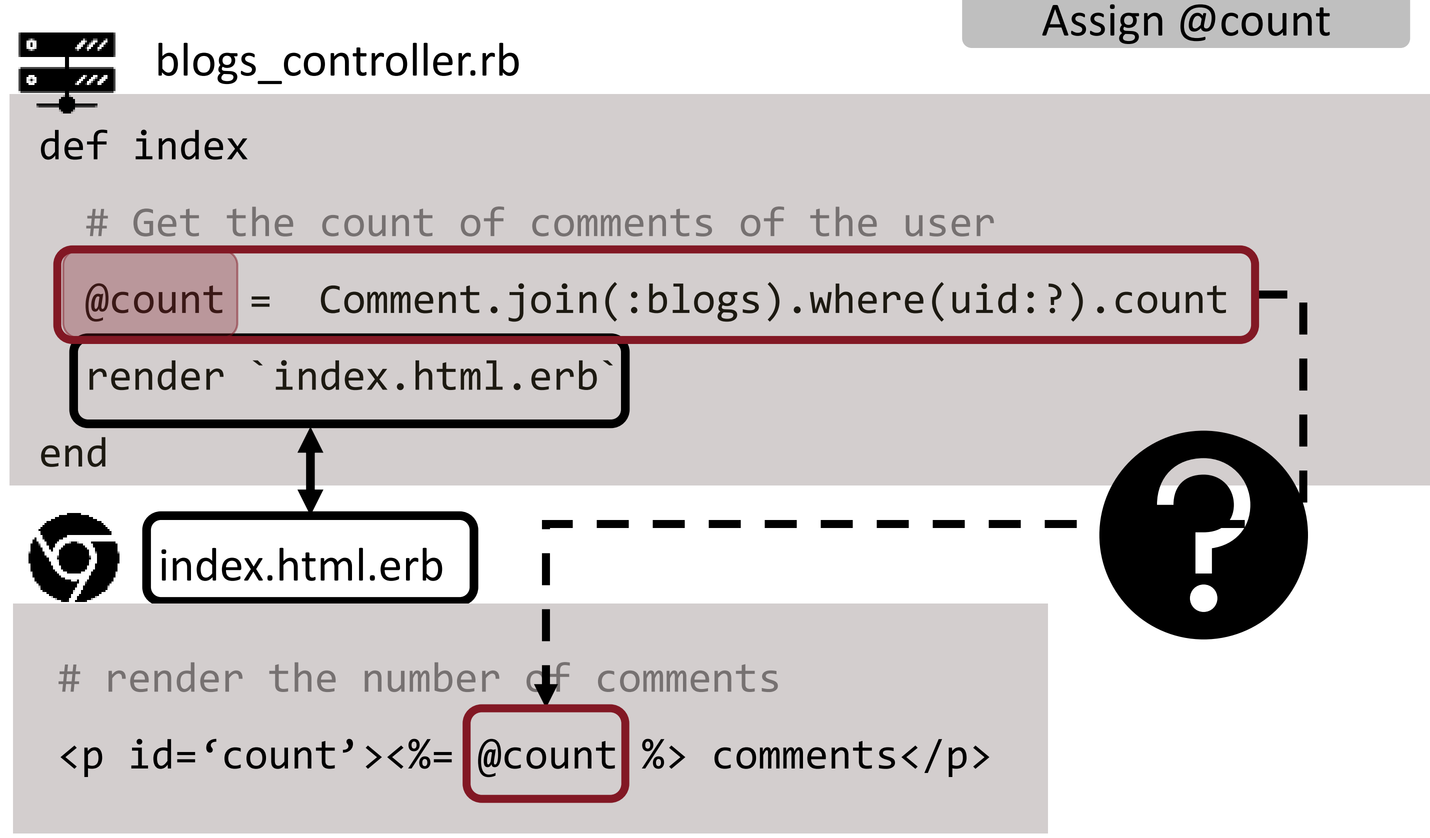


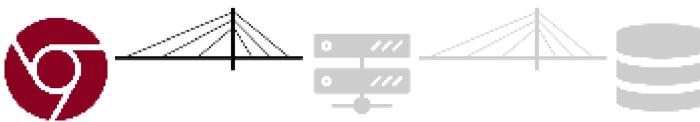
HOW1: Which application code is used to generate which HTML tag?

If it is inside one program, a data dependency analysis will be enough





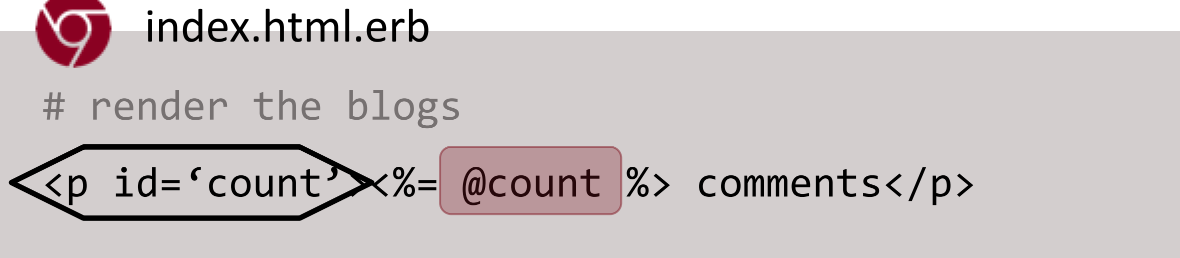




HOW1: Which application code is used to generate which HTML tag?

blogs_controller.rb

```
def index
  # Get the count of comments of the user
  @count =  Comment.join(:blogs).where(uid:?).count


  render `index.html.erb`
end
```

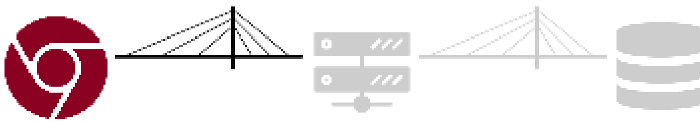


index.html.erb

```
# render the blogs
<p id='count'><%= @count %> comments</p>
```

@count

tag count



HOW1: Which application code is used to generate which HTML tag?

blogs_controller.rb

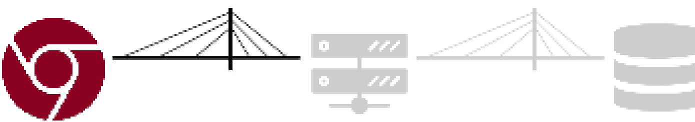
```
def index
  # Get the count of comments of the user
  @count =  Comment.join(:blogs).where(uid:?).count


  render `index.html.erb`
end
```

index.html.erb

```
# render the blogs
<p id='count'><%= @count %> comments</p>
```

@count

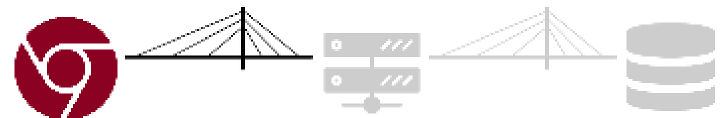
tag count



HOW1: Which application code is used to generate which HTML tag?

blogs_controller.rb

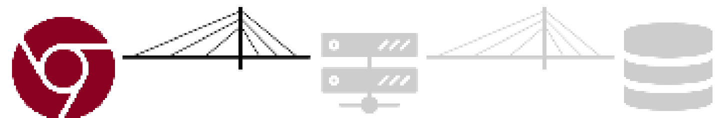
```
def index
  # Get the count of comments of the user
  @count =  Comment.join(:blogs).where(uid:?).count


  @count   tag count


end
```



Extend the program dependency graph with tag information

blogs_controller.rb

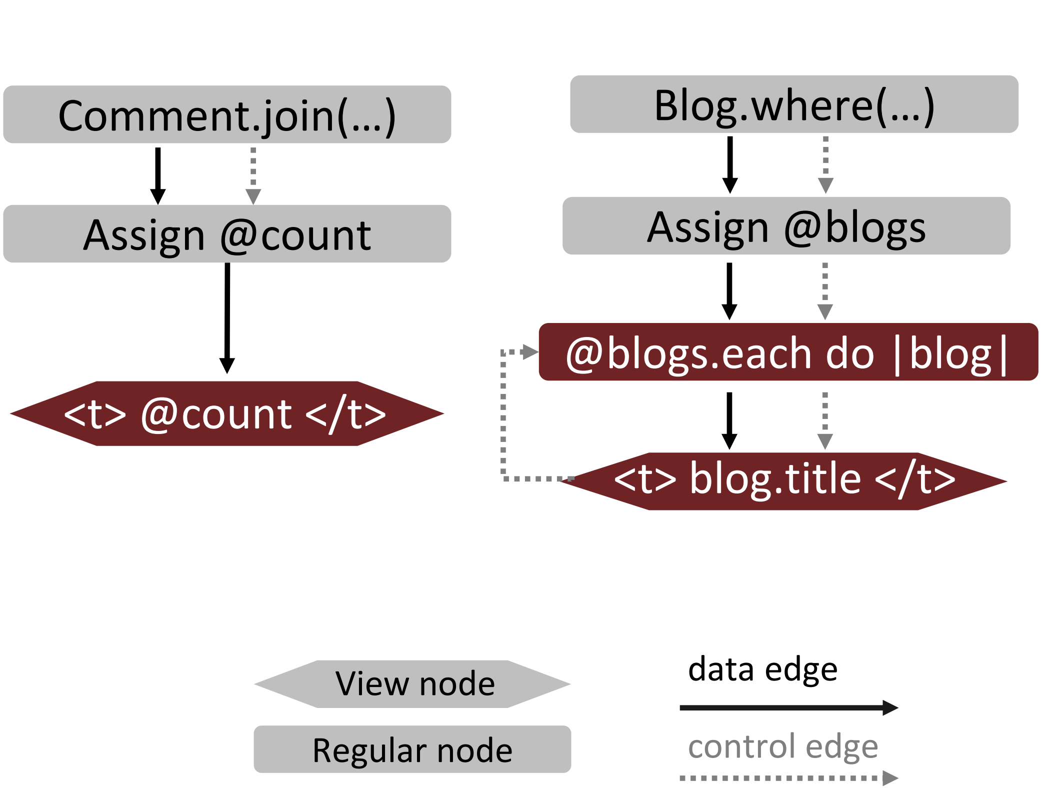
```
def index
  # Get the count of comments of the user
  @count =  Comment.join(:blogs).where(uid:?).count

  @count   tag count

end
```

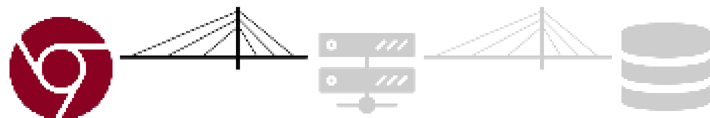


Extend the program dependency graph with tag information

blogs_controller.rb

```
def index
  # Get the count of comments of the user
  @count =  Comment.join(:blogs).where(uid:?).count
  # Get a user's blogs
  @blogs =  Blog.where(uid:?)
  @count        tag count
  @blogs.each do |blog|
    blog.title        tag blog
  end
end
```

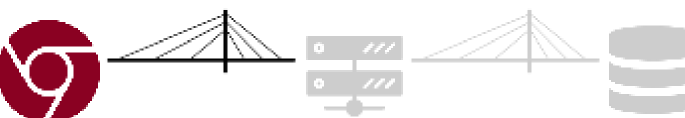




How to conduct refactoring?

- HOW1: Which application code is used to generate which HTML tag?
- HOW2: Is this statement mapped to a scaling query?
- HOW3: What is the complexity of the application code?

Extend Ruby compiler to understand selected HTML/ORM/SQL information

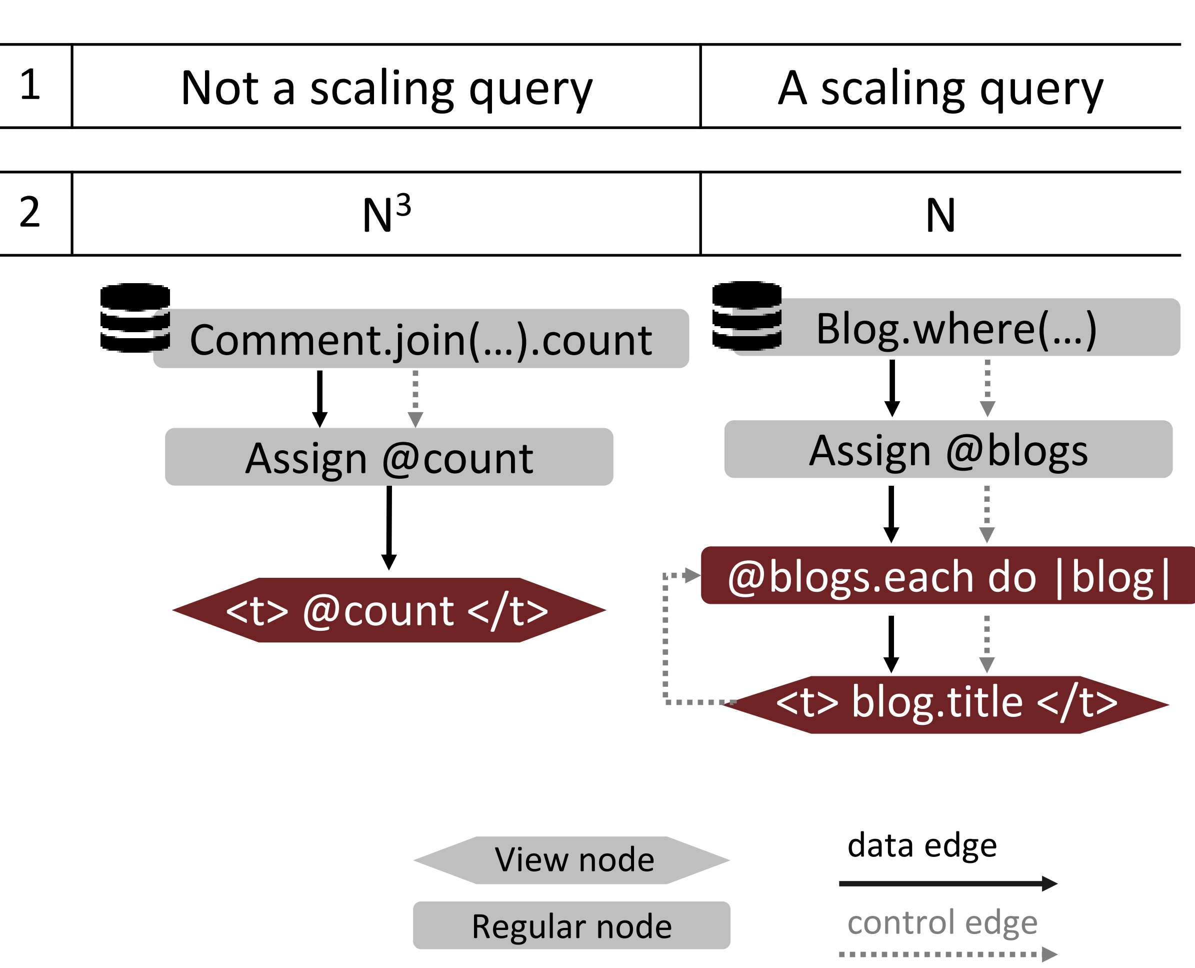


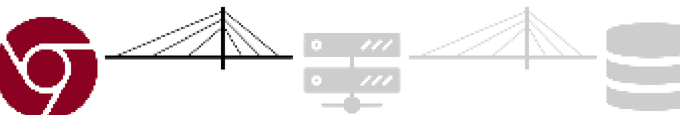
Extend the program dependency graph with query information

blogs_controller.rb

```
def index
  # Get the count of comments of the user
  Comment.join(:blogs).where(uid:?).count
  # Get a user's blogs
  Blog.where(uid:?)
end
```

1. Will it return scaling query results
   No aggregation, no limit keyword
2. Query cost information
   join: N^2, **where**: N if no index …

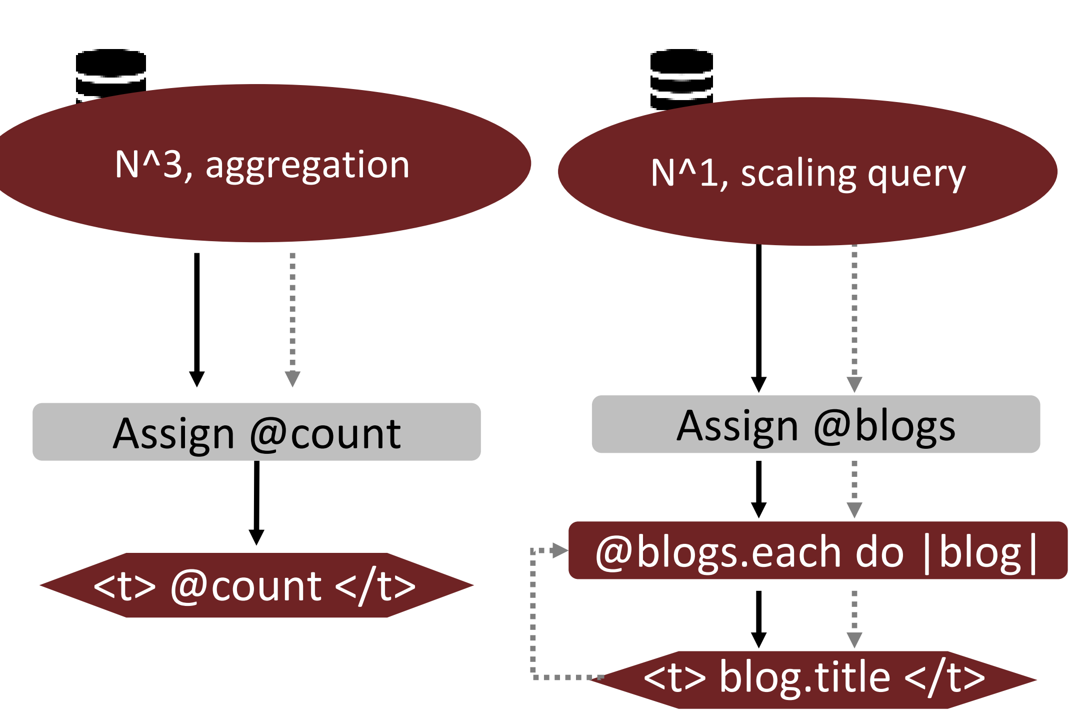




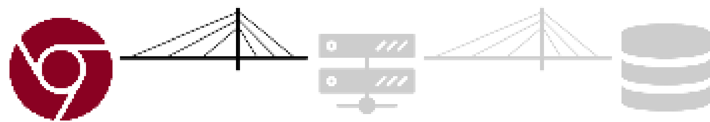
Extend the program dependency graph with query information

blogs_controller.rb

```
def index
  # Get the count of comments of the user
  @count =  Comment.join(:blogs).where(uid:?).count
  # Get a user’s blogs
  @blogs =  Blog.where(uid:?)
end
```



1. Will it return scaling query results
   No aggregation, no limit keyword
2. Query cost information
   join: N^2, **where**: N^1 if no index …



Automatic fixing for pagination

controllers/blogs_controller.rb

```
# paginate the query
- @blogs =  Blog.where(uid:?).order(:created)
+ @blogs =  Blog.where(uid:?).order(:created).paginate()
```

views/blogs/index.html.erb

```
# create page navigation bar
+ <%= will_paginate @blogs %>
```

Automatic fixing for asynchronously loading

views/blogs/_comment_cnt.html.erb

```
# create extra view
+ <span> <%= @coun
```

config/routes.rb

```
# set routes
+ get :comment_cnt, :controller=>blogs
```

views/blogs/index.html.erb

```
# replacing original span
-  <span> <%= @count %></span>
-  + <%= render_async _comment_cnt_path %>
```

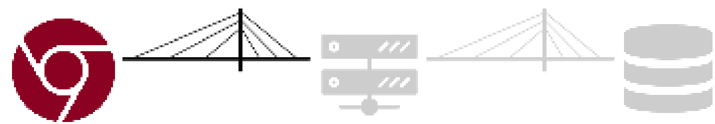
contronllers/blogs_controller.rb

```
# checking if blog exists
+ def comment_cnt
+ @count = Comments….count
+ render :partial => ‘comment_cnt’
+ end
```

Applications.html.er

```
# set application
+ <% content_for :render_async %>
```

Not for reading

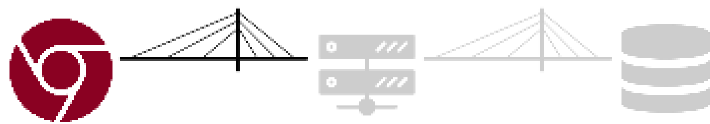


Evaluation results

149 optimization opportunities on 12 apps.

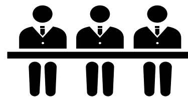
Speed up for 15 sampled opportunities.

	End-to-end	Server
Average	4.5X	8.6X
Max	17.2X	37.8X



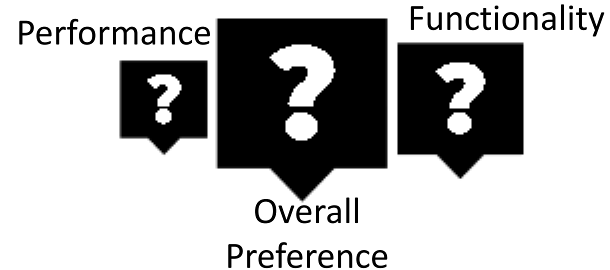
Evaluation results

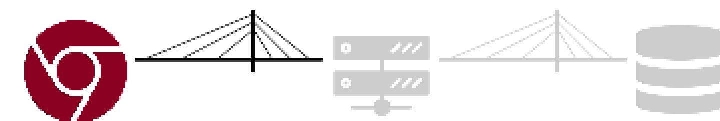
User Study on View Design

100 participants from Amazon MTurk. 

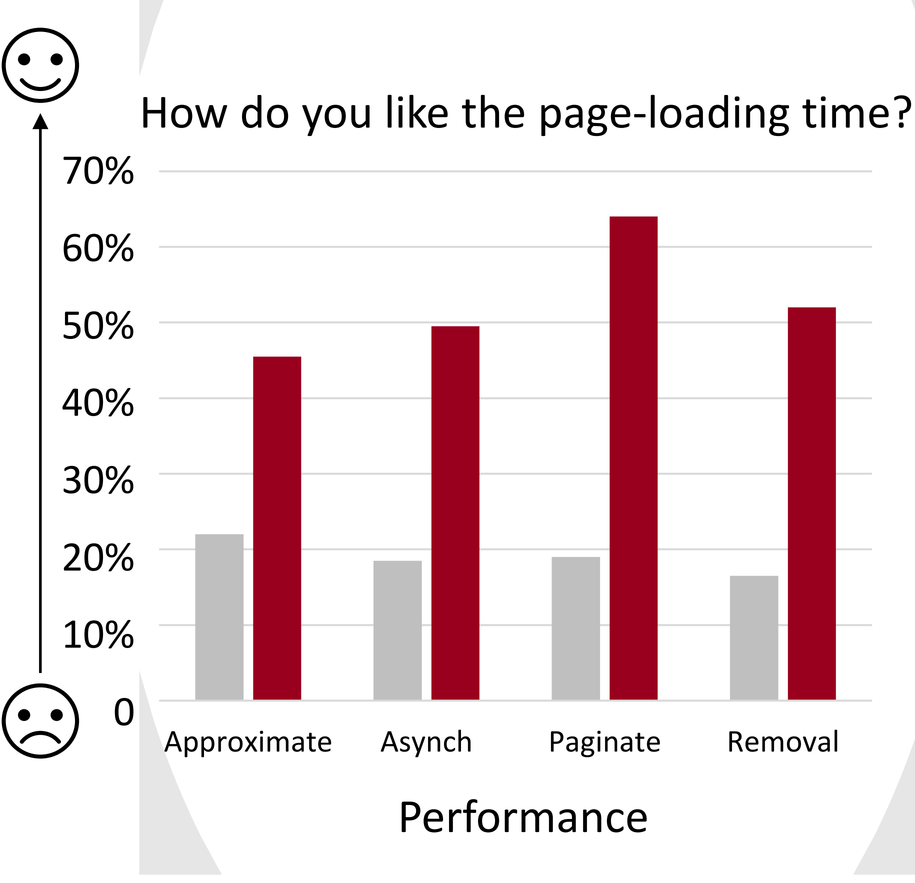
8 groups of pages, 1.5s diff in load time.

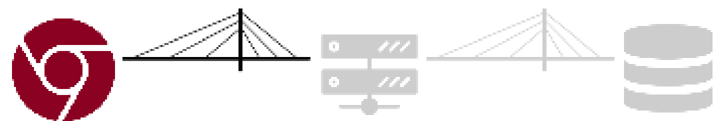
View the page and answer 3 questions:





Evaluation results





Evaluation results

How do you like the page-loading time?

How do you like the content rendered?

70%
60%
50%
40%
30%
20%
10%
0

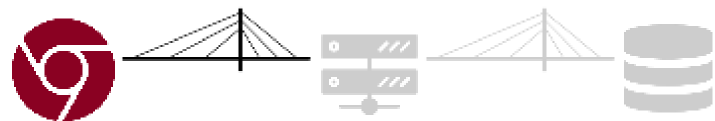
Approximate Asynch Paginate Removal

Performance

Approximate Asynch Paginate Removal

Functionality

Prefer original page
Prefer new page



Evaluation results

How do you like the page-loading time? | How do you like the content rendered? | Which web page do you like better?

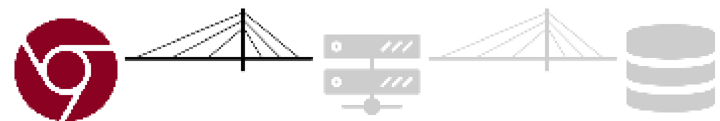
70%
60%
50%
40%
30%
20%
10%
0

Approximate Asynch Paginate Removal
Performance

Approximate Asynch Paginate Removal
Functionality

Approximate Asynch Paginate Removal
Overall

Prefer original page
Prefer new page



Evaluation results

How do you like the page-loading time? | How do you like the content rendered? | Which web page do you like better?

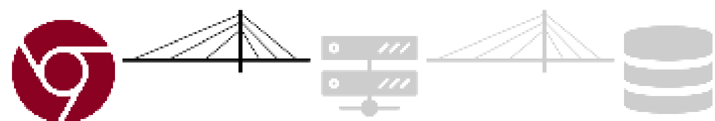
70%
60%
50%
40%
30%
20%
10%
0

Approximate Asynch Paginate Removal
Performance

Approximate Asynch Paginate Removal
Functionality

Approximate Asynch Paginate Removal
Overall

Prefer original page
Prefer new page



Evaluation results

How do you like the page-loading time? | How do you like the content rendered? | Which web page do you like better?

70%
60%
50%
40%
30%
20%
10%
0

Approximate | Asynch | Paginate | Removal
Performance

Approximate | Asynch | Paginate | Removal
Functionality

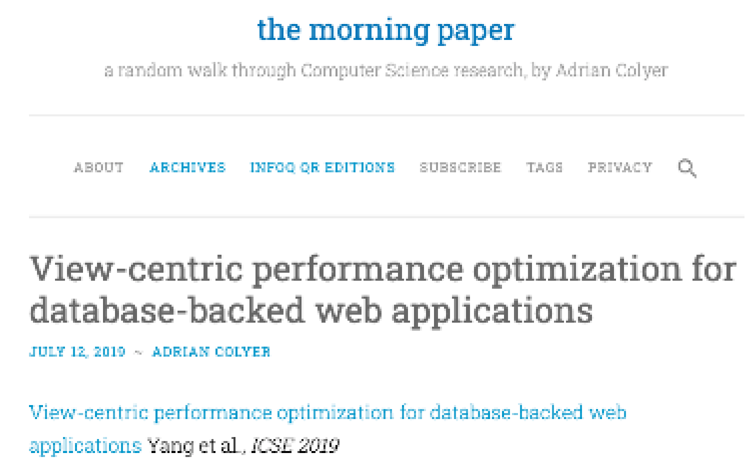
Approximate | Asynch | Paginate | Removal
Overall

Prefer original page | Prefer new page

- First work that conducting user-interface optimization
- 100+ user-interface optimization detected and refactored

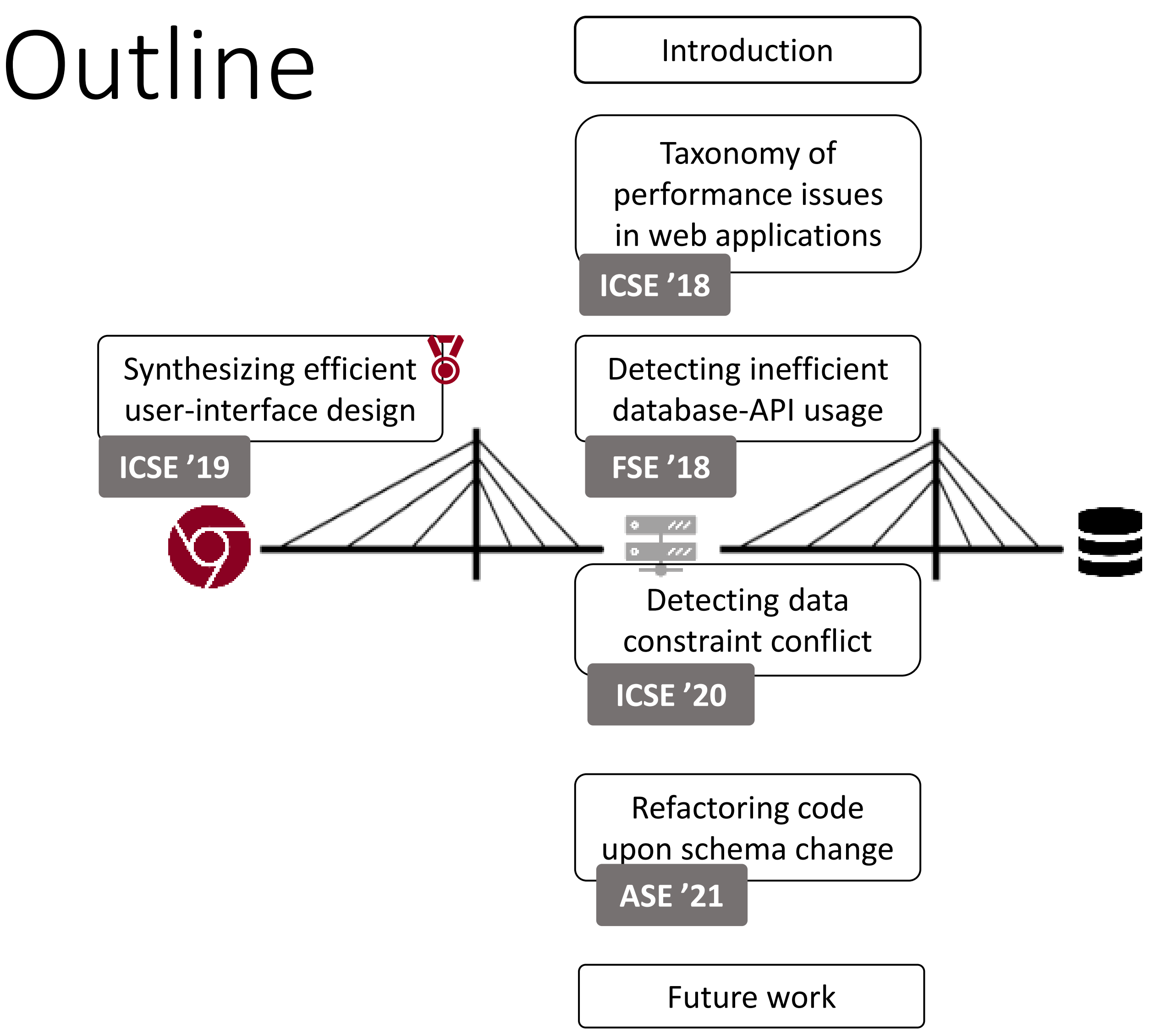
IMPACT

- Featured on morning paper
- Winning distinguished paper award



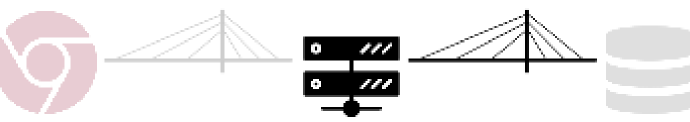
View-centric performance optimization, **ICSE '19**

Outline



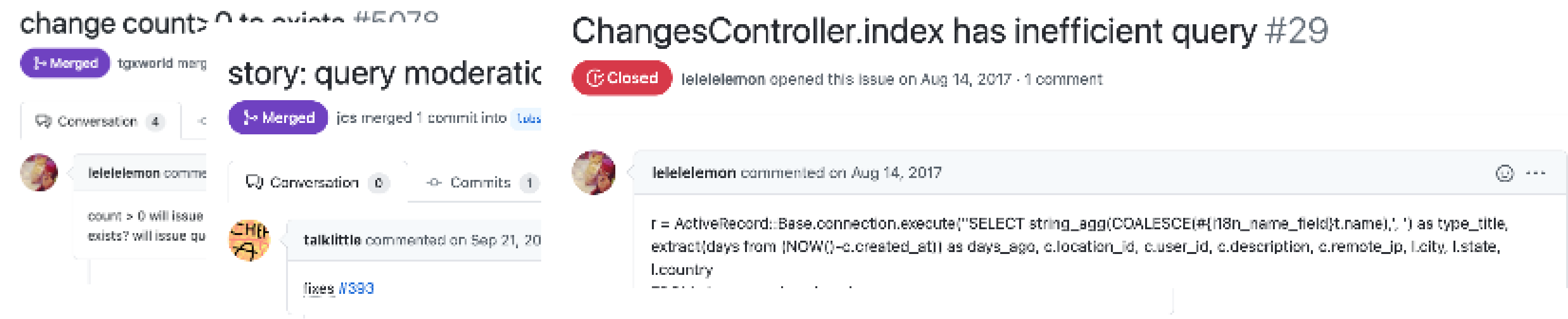
Performance

Correctness

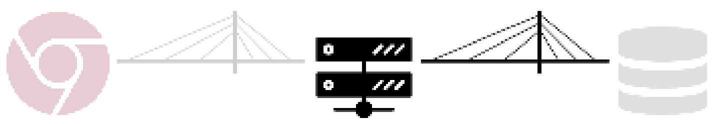


Detecting and fixing inefficient use of ORM APIs

- **Why?**
  - 51% of performance issues are caused by inefficient use of ORM APIs



Powerstation: Automatically detecting and fixing inefficiencies of database-backed web applications in ide. FSE '18
Yang Junwen, Cong Yan, Pranav Subramaniam, Shan Lu, and Alvin Cheung.



Examples

Overly expensive query!

```
# checking if blog exists
- if Blog.present?
+ if Blog.exists?
  …
end
```

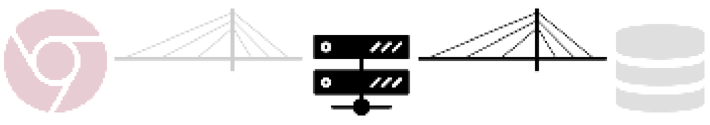
Redundant queries!

```
# filter blogs in blocklist
+ blocks = BlockList.all
blogs.reject do |b|
- b.title in BlockList.all
+ b.title in blocks
end
```

Query result is not used!

```
# load user from DB
- user.reload
  … # no reference to user
user.reload
```

select * from blocklists



Challenges

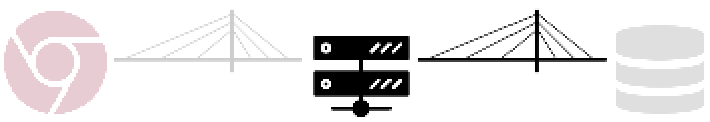
1. Existing compilers do not understand ORM APIs and SQL

```
# checking if blog exists
if Blog.present?
  …
end
```

```
# filter blogs in blocklist
blogs.reject do |b|
b.title in BlockList.all
end
```

```
# load user from DB
user.reload
  … # no reference to user
user.reload
```


Detecting inefficient database-API usage, **FSE ’18**



Challenges

1. Existing compilers do not understand ORM APIs and SQL

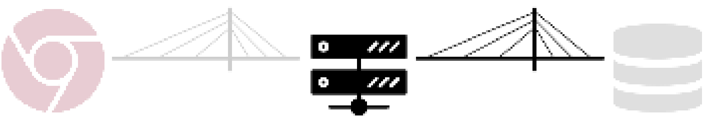
2. Wide varieties of inefficiencies associated with ORM APIs

Build one tool for each API mis-use?

```
# checking if blog exists
if Blog.present?
  …
end
```

```
# filter blogs in blocklist
blogs.reject do |b|
b.title in BlockList.all
end
```

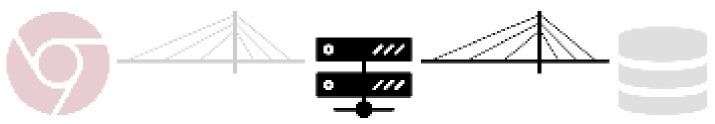
```
# load user from DB
user.reload
  … # no reference to user
user.reload
```



Solutions

1. Existing compilers do not understand ORM APIs and SQL

   Solved by our database-aware program dependency graph



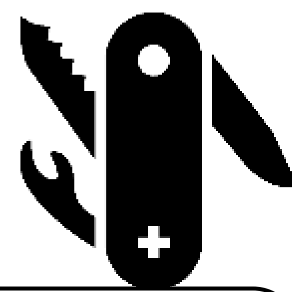
Solutions

1. Existing compilers do not understand ORM APIs and SQL

   Solved by our database-aware program dependency graph

2. Wide varieties of inefficiencies associated with ORM APIs

   Leveraging traditional compiler optimization algorithms



```
# checking if blog exists
- if Blog.present?
+ if Blog.exists?
  …
end
```

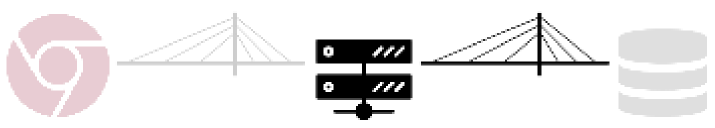
Strength reduction

```
# filter blogs in blocklist
+ blocks = BlockList.all
blogs.reject do |b|
- b.title in BlockList.all
+ b.title in blocks
end
```

Loop-invariant motion

```
# load user from DB
- user.reload
  … # no reference to user
user.reload
```

Dead code elimination

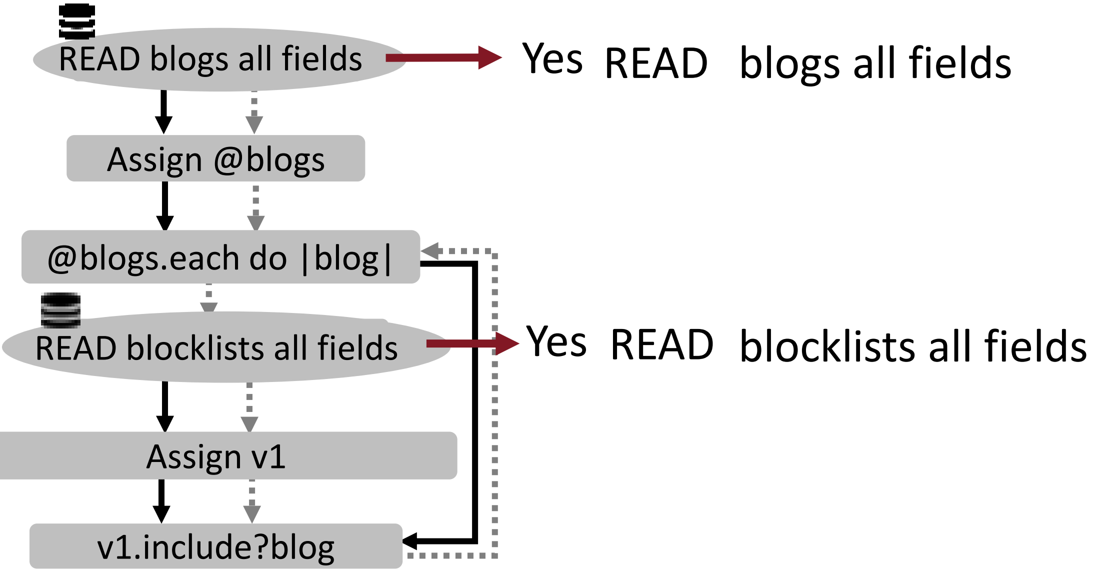


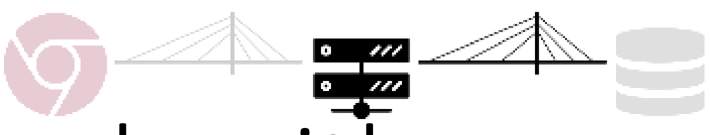
Cross-stack analysis

Extending the traditional program dependency graph with selected query information:

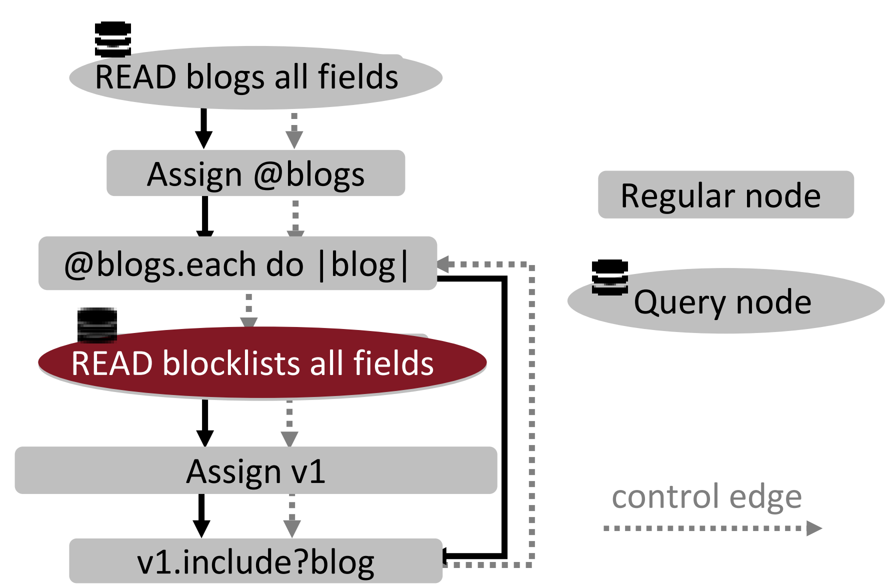
- Will this ORM API be translated to a SQL query
- Is it a READ or a WRITE query
- What database table fields are accessed

```
# filter blogs in blocklist
blogs.reject do |b|
b.title in BlockList.undeleted
end
```





Leveraging traditional optimization algorithms



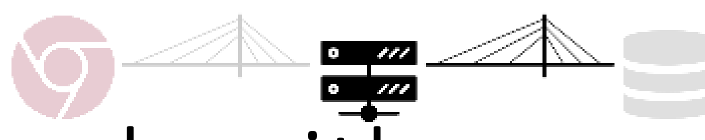
Taking loop-invariant query as an example

Detection

- Locating query node inside one loop
- Checking whether it depends on nodes inside loop

Fixing

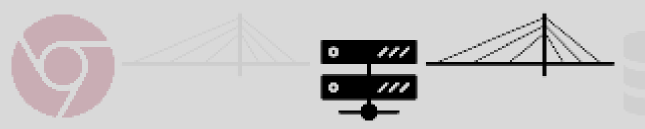
- Hoisting the query outside the loop



Leveraging traditional optimization algorithms

	Previous work			Our tool
Anti-pattern	P1	P2	P3	PowerStation
Loop-invariant query				Detection, Fixing
Dead-store query				Detection, Fixing
Common sub-expr query		Detection		Detection
API misuses			Detection	Detection, Fixing
Unused data-retrieval query	Detection			Detection, Fixing
Inefficient rendering				Detection, Fixing

Detecting inefficient database-API usage, **FSE ’18**



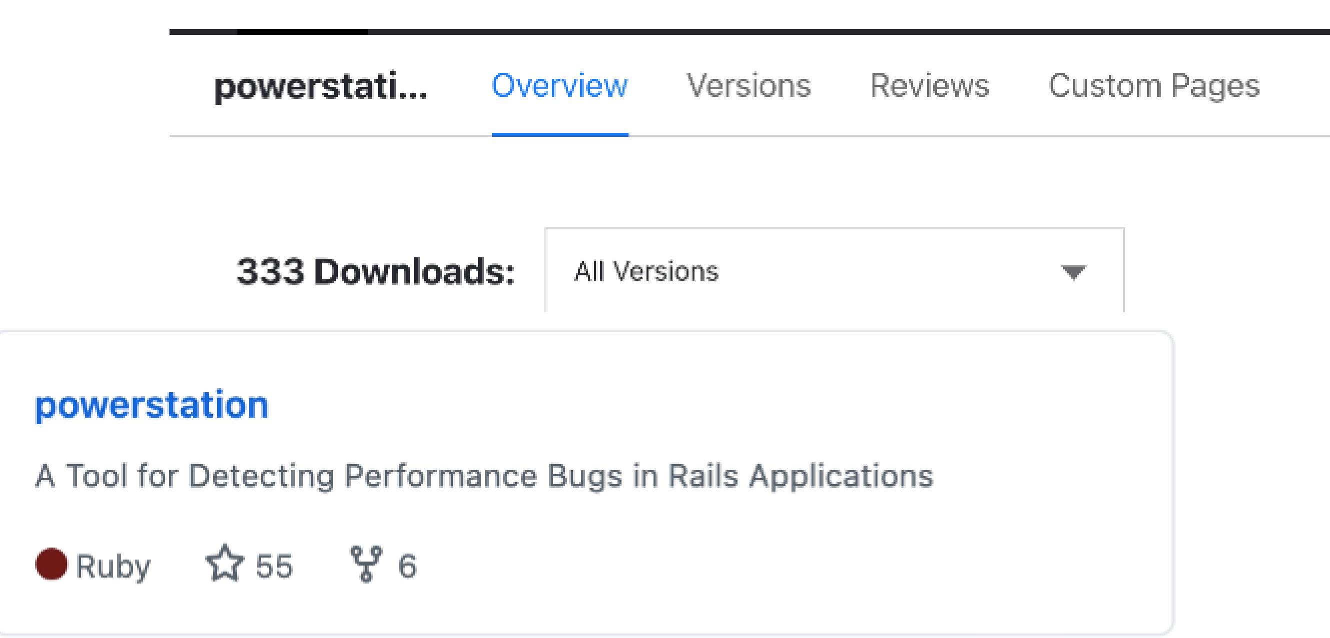
Detected thousands of

unknow bugs

and fixed hundreds of them

Plugin downloaded more

than 300 times

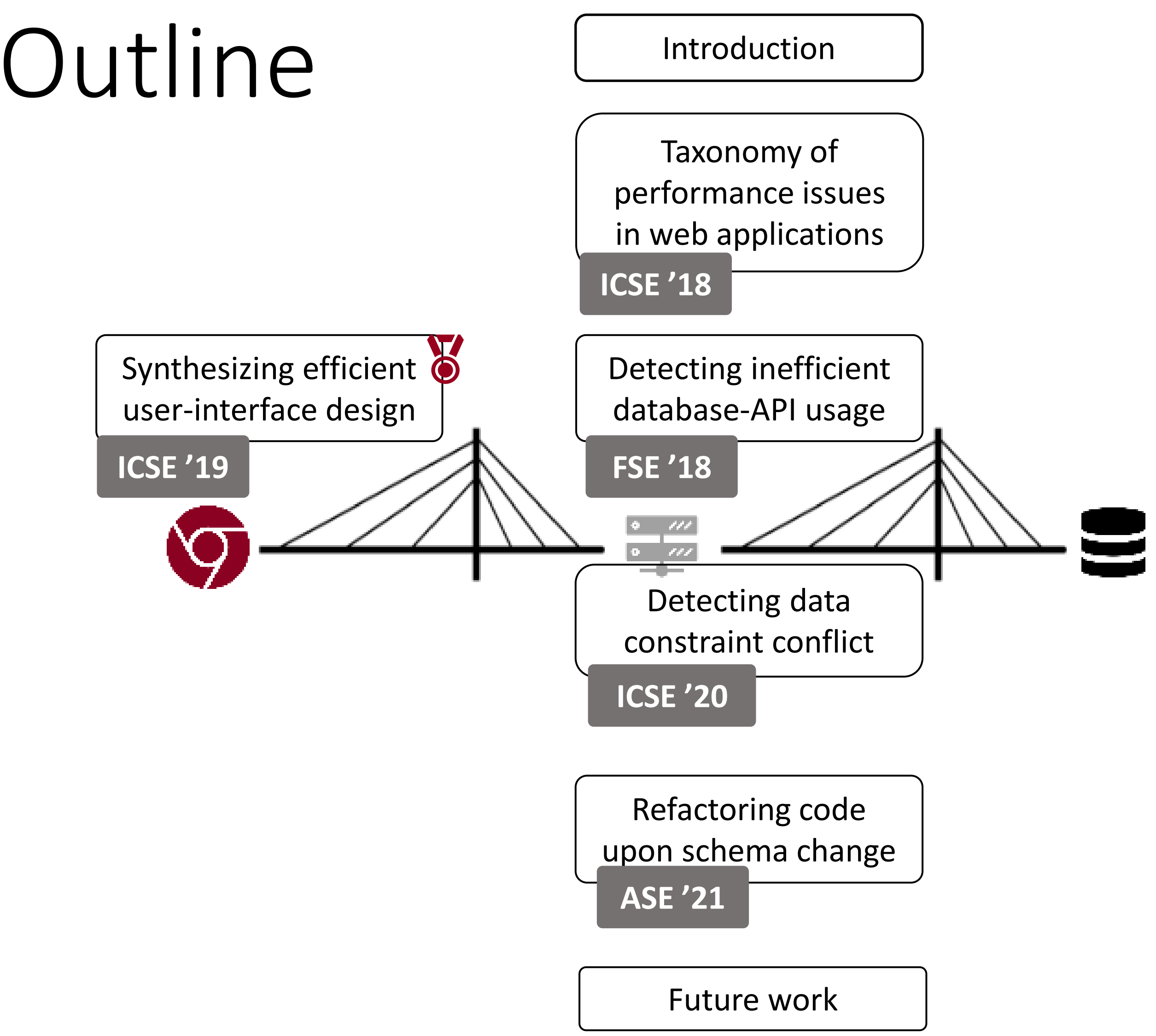


Summary of performance problems

- Cross-stack optimization is the key technique behind
  - It’s not to extend one stack to fully understand another,
    but use selective information
- User-interface optimization is important
  - Performance unfriendly interface cannot be solved by traditional optimizer

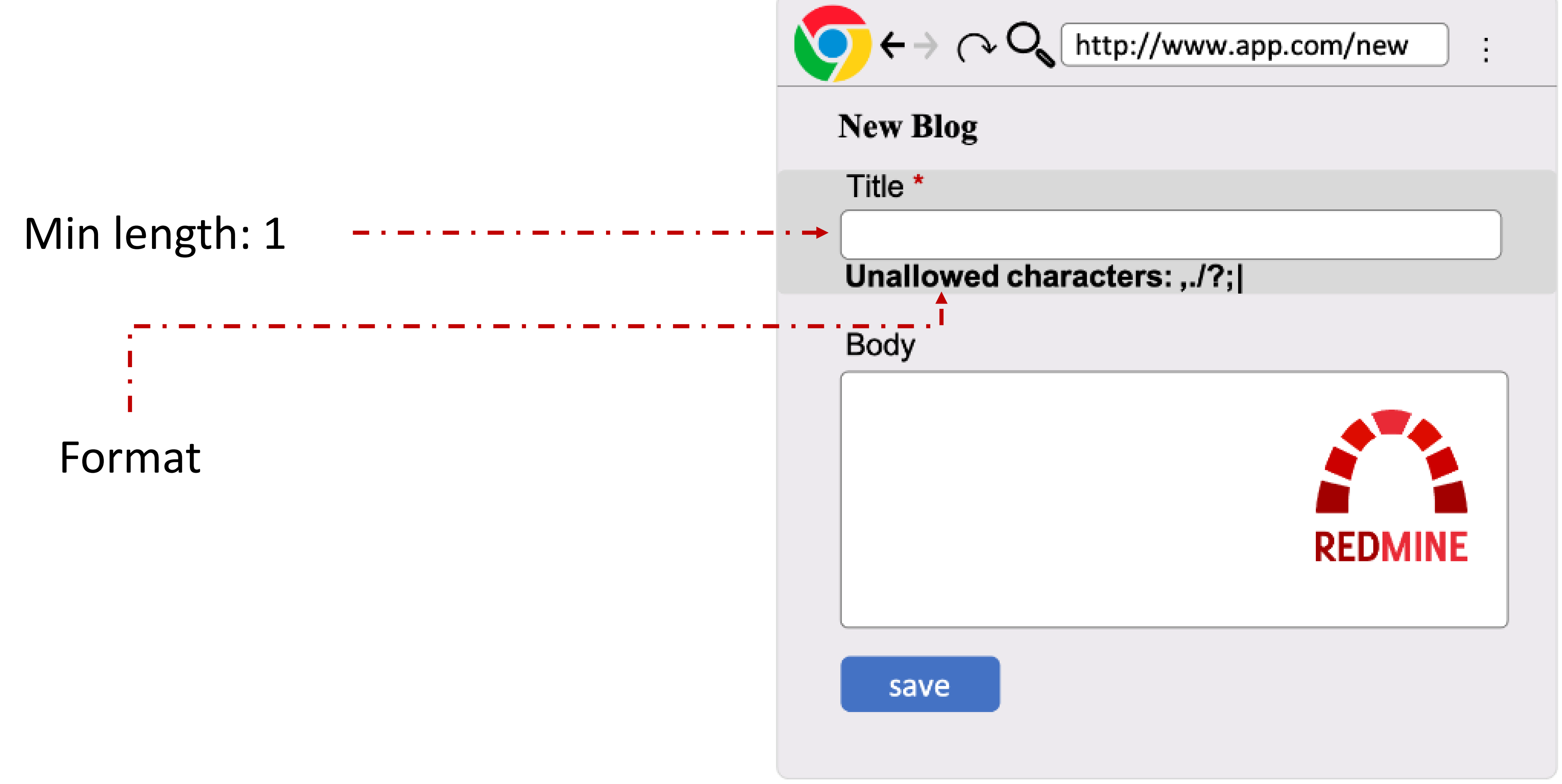
Performance

Outline



Correctness

Data constraints



Data constraints in web applications

- Large amount of data with many constraints
- Data checked across multiple stacks
- Frequent upgrades and migrations

77% fields with constraints
1.4 constraints per field

Field	Type	Null	...
title	varchar(60)	YES	...

Field	Type	Null	...
title	varchar(30)	NO	...

```
<input pattern=`.+` />
```

```
validates_length: title, max: 60,
message: 'title is too long'
```

```
CREATE TABLE blogs ( title
VARCHAR(30))
```

Managing data constraints in database-backed web applications, **ICSE '20**

Detecting real-world constraint problems

Inconsistency across stacks

- 200+ fields forbidden … but null by default in DB
- 88 fields required to be unique in app …
- 57 in(ex)clusion constraints specified in app, but missed in DB
- 133 conflicting length/numericality constraints between app and DB

Page crash

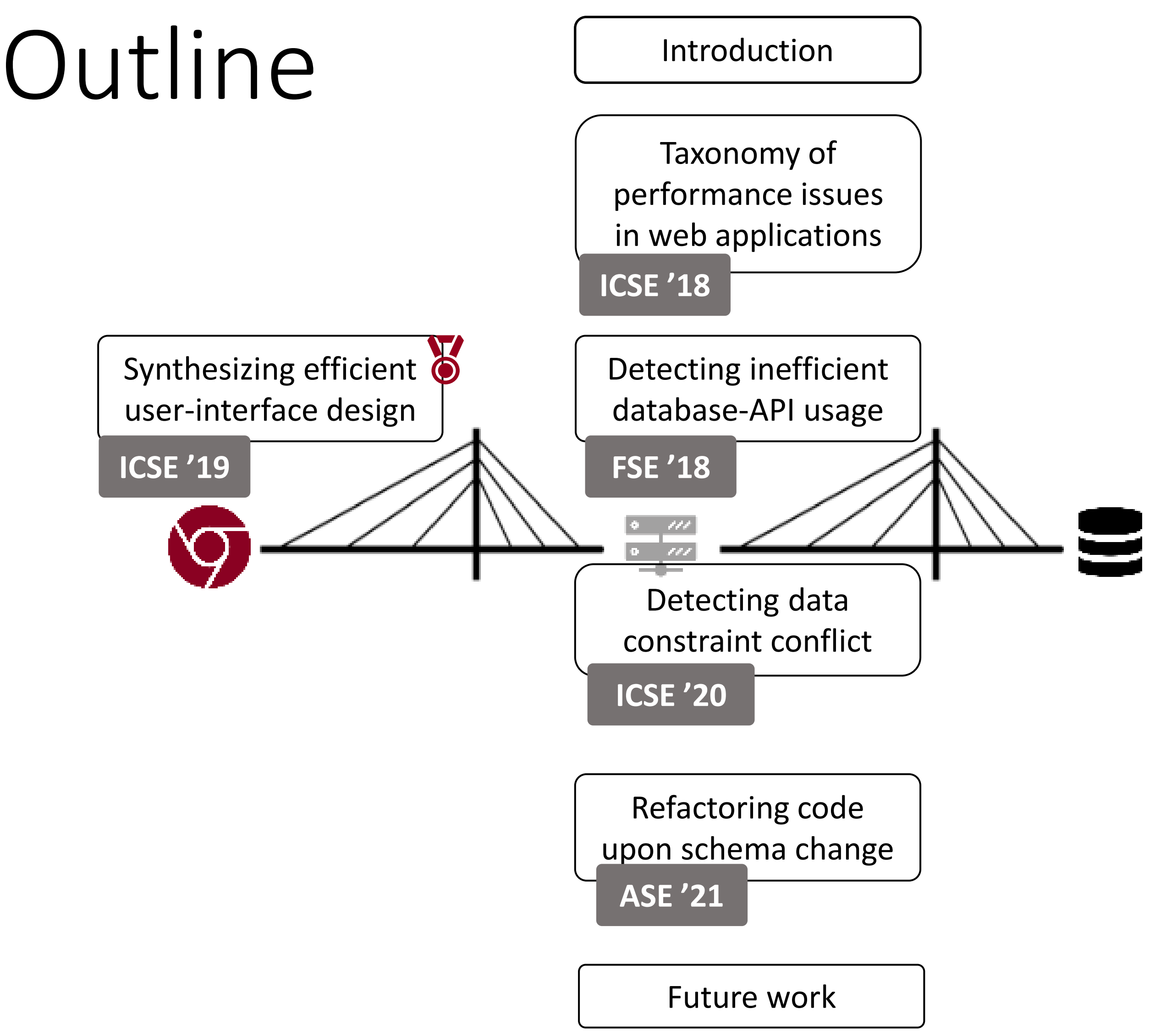
Inconsistency …

- > 25% of changes tighten constraints … ata fields

Upgrade failure

Performance

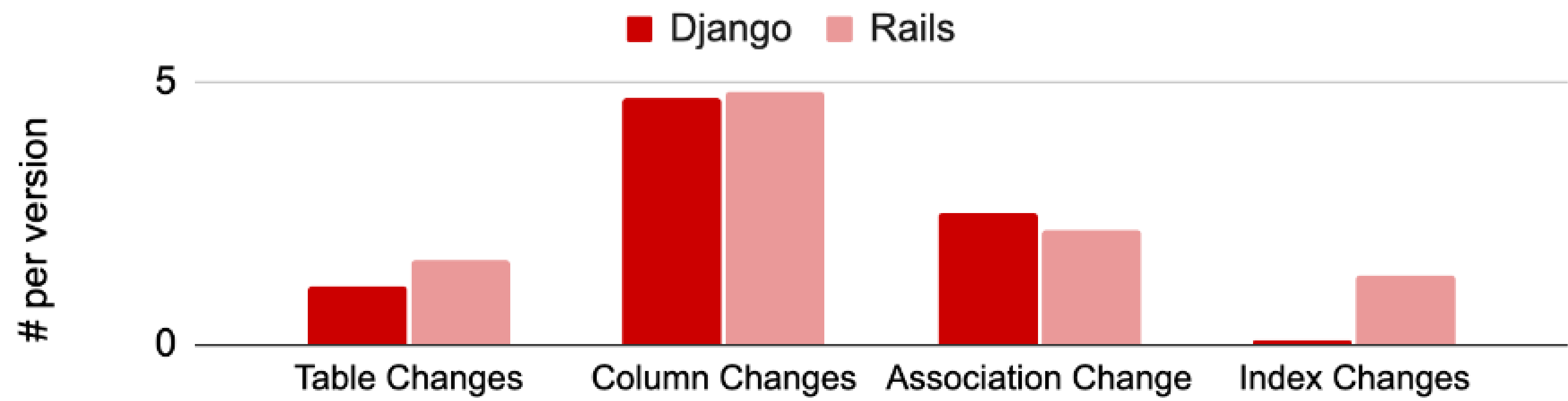
Outline



Correctness

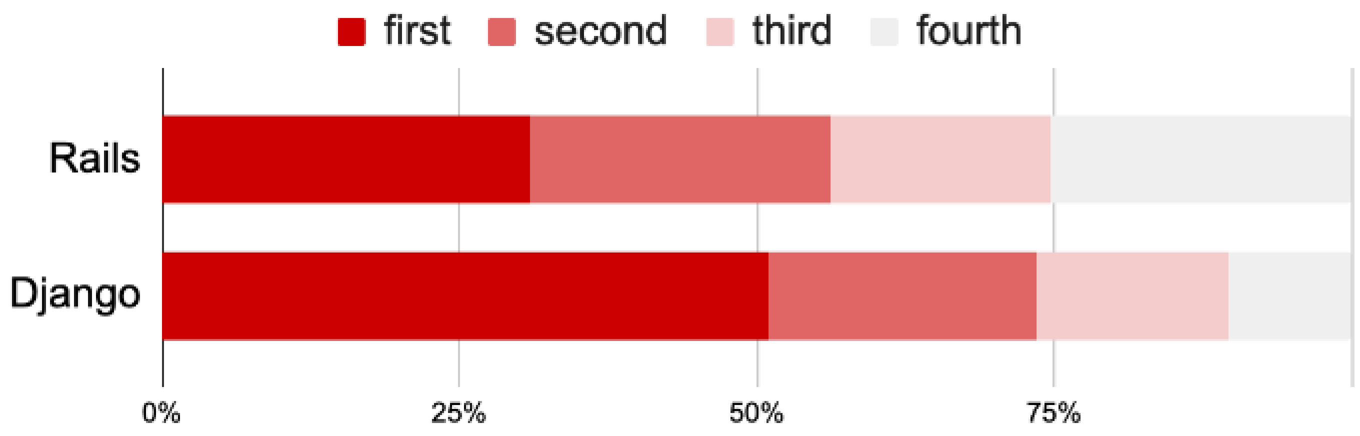
Schema changes in web applications

- 18 ~ 85% of application versions contain at least one schema change
- Changes to various aspects of the schema are all common



Schema changes in web applications

- 18 ~ 85% of application versions contain at least one schema change
- Changes to various aspects of the schema are all common
- Changes are across the development history

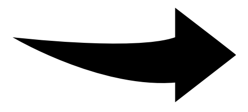


Automated Code Refactoring upon Database-Schema Changes in Web Applications, **ASE ’21**

Schema change requires app code change

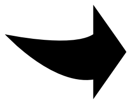
Field	Type	Null	Default	…
title	varchar(30)	NO	NULL	…

```
rename_column
    :blogs, :title, :header
```



Field	Type	Null	Default	…
header	varchar(30)	NO	NULL	…

```
blog = Blog.where(“title = ?”)
blog.title
```



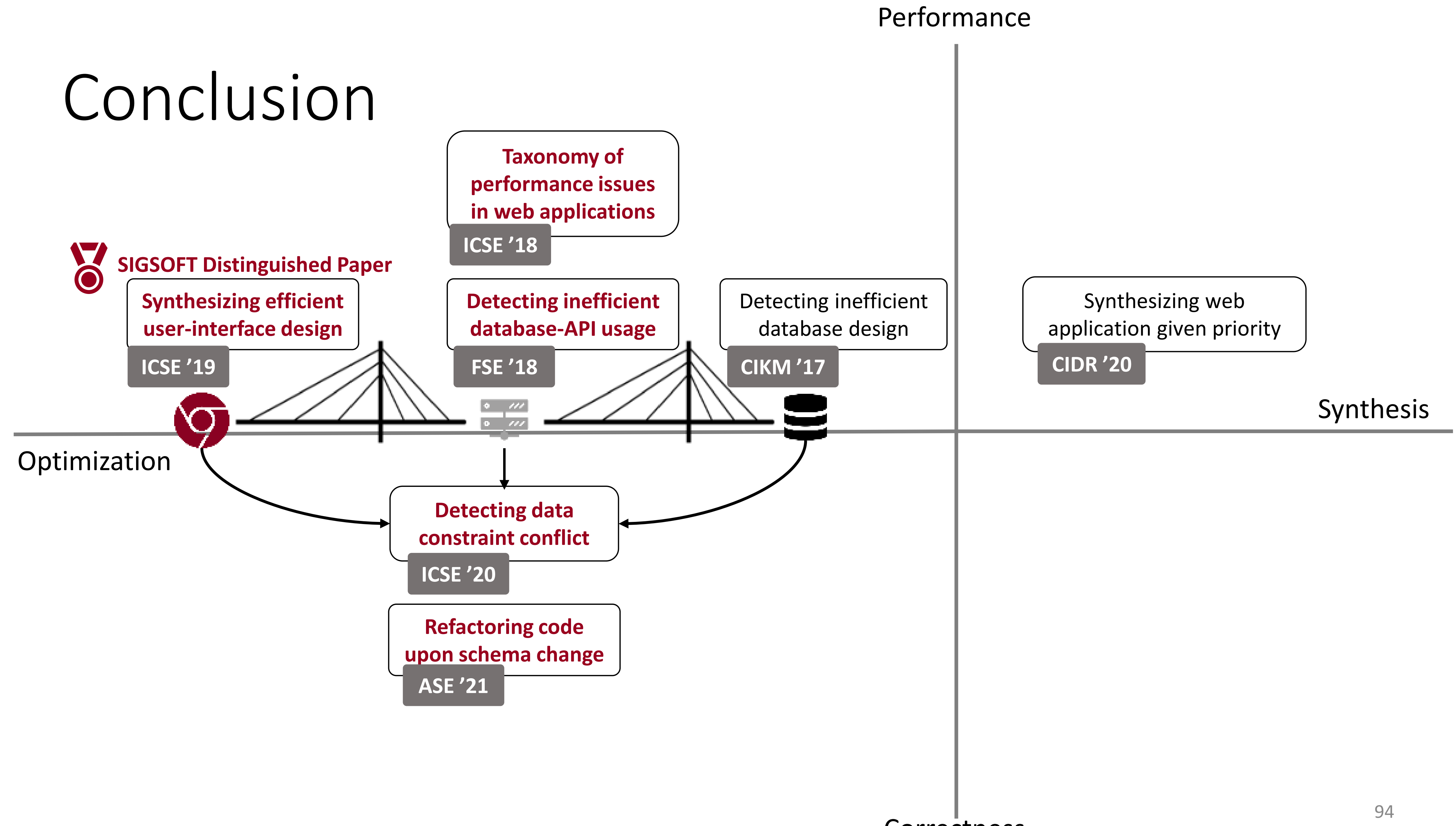
```
blog = Blog.where(“header = ?”)
blog.header
```

Approach

- Extract schema change from migration files
- Extract queries using database aware program dependency graph
- Inconsistency detection and refactoring suggestion

Evaluation result

	Rails	Django	Total
# of inconsistent queries	38	48	86
# existing in release	20	10	30
# of inconsistent queries in latest versions	1	10	11


Performance
Conclusion
Taxonomy of performance issues in web applications
ICSE ’18
SIGSOFT Distinguished Paper
Synthesizing efficient user-interface design
ICSE ’19
Detecting inefficient database-API usage
FSE ’18
Detecting inefficient database design
CIKM ’17
Synthesizing web application given priority
CIDR ’20
Synthesis
Optimization
Detecting data constraint conflict
ICSE ’20
Refactoring code upon schema change
ASE ’21
Correctness
94

Conclusion

- Why were there so many problems?
  - The huge gaps among web users, web apps, and DB engines
  - The large scale of modern systems

- What is the solution?
  - Synthesis?
  - Better testing?