Updated 1/14/2023

Topic 1: Data models and query languages
Unit 1: SQL (continued)
Lecture 2

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp23)

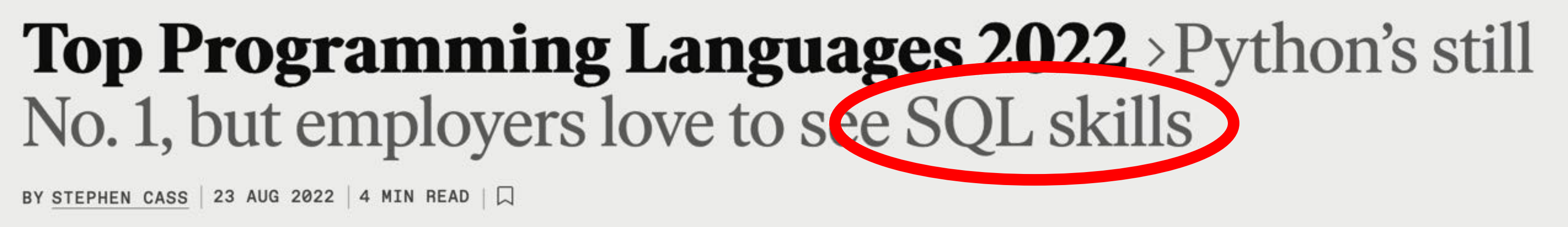
https://northeastern-datalab.github.io/cs7240/sp23/

1/13/2023

Pre-class conversations

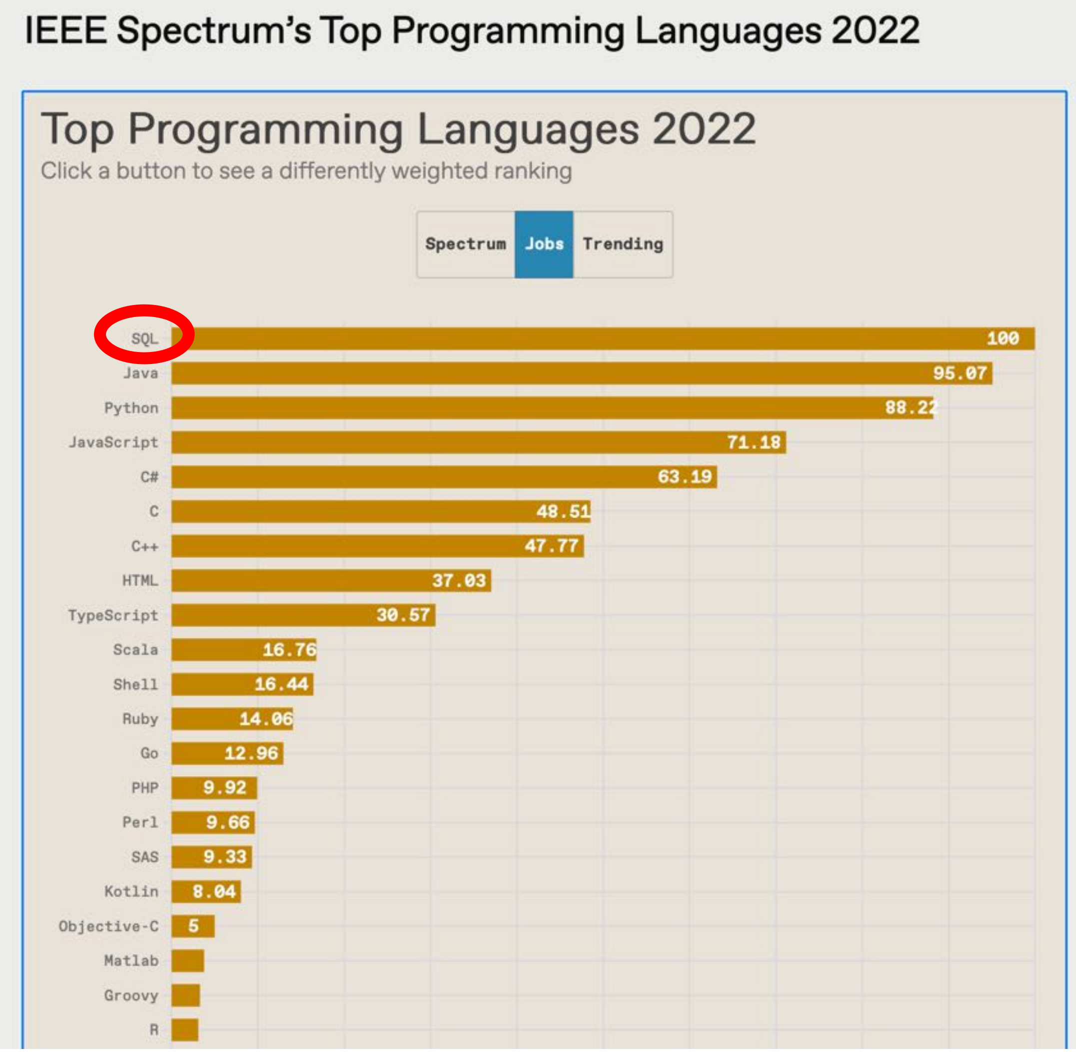
- Last class summary
- Class procedures based on past suggestions:
    - Secondary posting of class scribes to Piazza (optionally anonymous). I will comment on both Canvas and Piazza
    - Already installed Postgres?
    - The possible downsides of homeworks with self-determined deadlines: you are in charge
    - Interactivity in class

- Today:
    - SQL continued

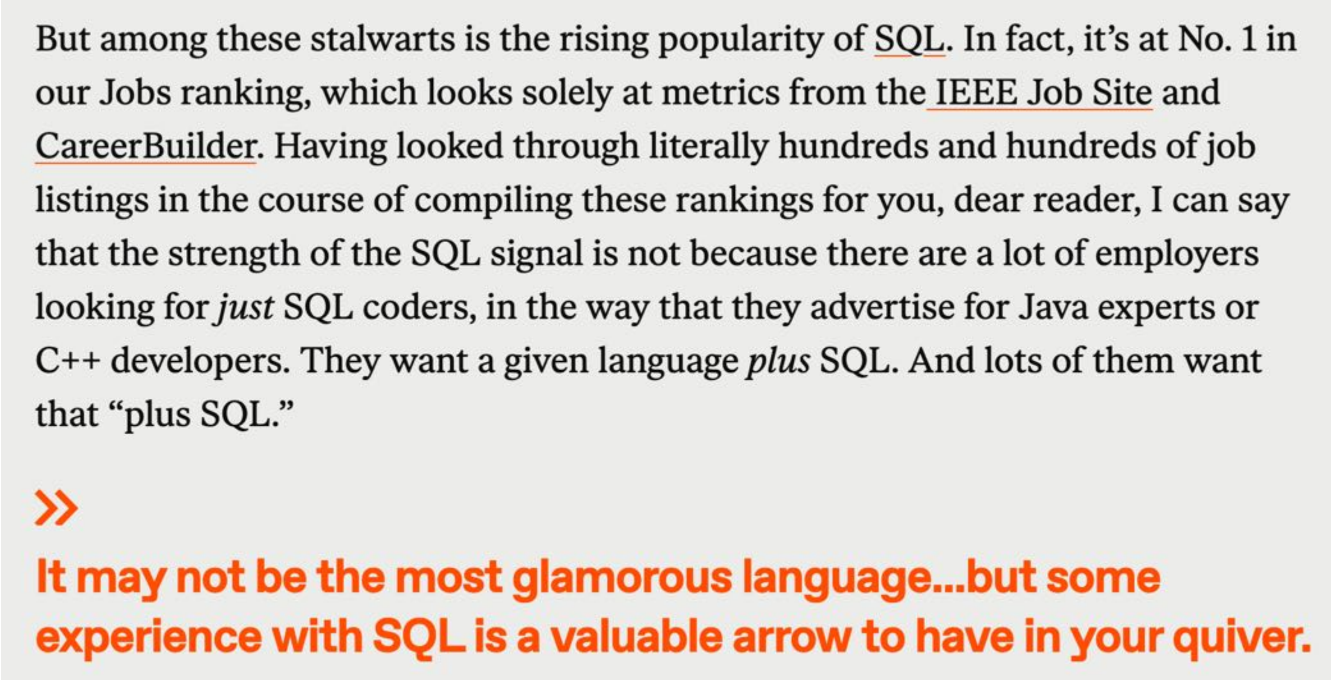


Top Programming Languages 2022 › Python's still No. 1, but employers love to see SQL skills

BY STEPHEN CASS | 23 AUG 2022 | 4 MIN READ



IEEE Spectrum's Top Programming Languages 2022



But among these stalwarts is the rising popularity of SQL. In fact, it's at No. 1 in our Jobs ranking, which looks solely at metrics from the IEEE Job Site and CareerBuilder. Having looked through literally hundreds and hundreds of job listings in the course of compiling these rankings for you, dear reader, I can say that the strength of the SQL signal is not because there are a lot of employers looking for *just* SQL coders, in the way that they advertise for Java experts or C++ developers. They want a given language *plus* SQL. And lots of them want that "plus SQL."

»

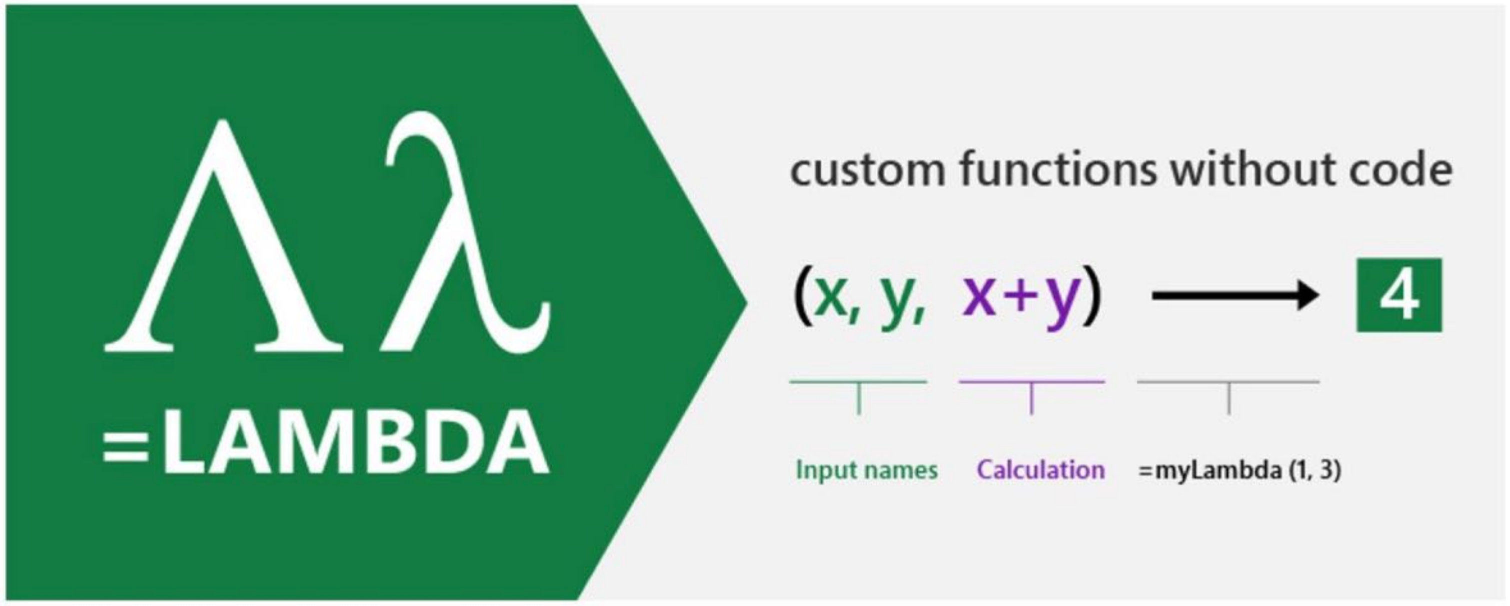
It may not be the most glamorous language...but some experience with SQL is a valuable arrow to have in your quiver.

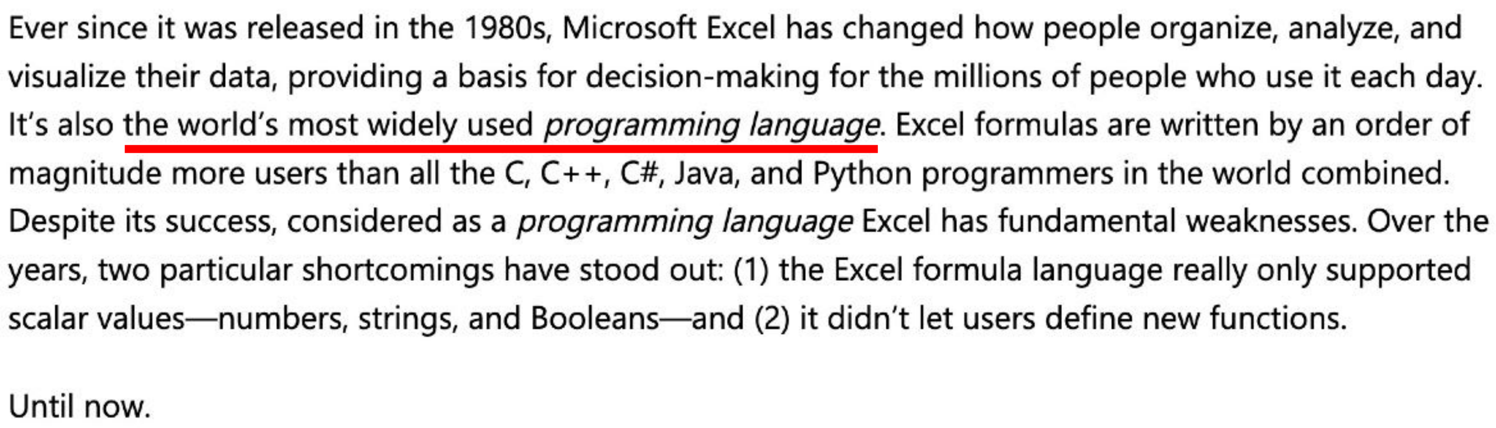
Source: https://spectrum.ieee.org/top-programming-languages-2022

Fun question: What is the most popular PL?

?

Fun question: What is the most popular PL?




Ever since it was released in the 1980s, Microsoft Excel has changed how people organize, analyze, and visualize their data, providing a basis for decision-making for the millions of people who use it each day. It's also the world's most widely used *programming language*. Excel formulas are written by an order of magnitude more users than all the C, C++, C#, Java, and Python programmers in the world combined. Despite its success, considered as a *programming language* Excel has fundamental weaknesses. Over the years, two particular shortcomings have stood out: (1) the Excel formula language really only supported scalar values—numbers, strings, and Booleans—and (2) it didn't let users define new functions.

Until now.

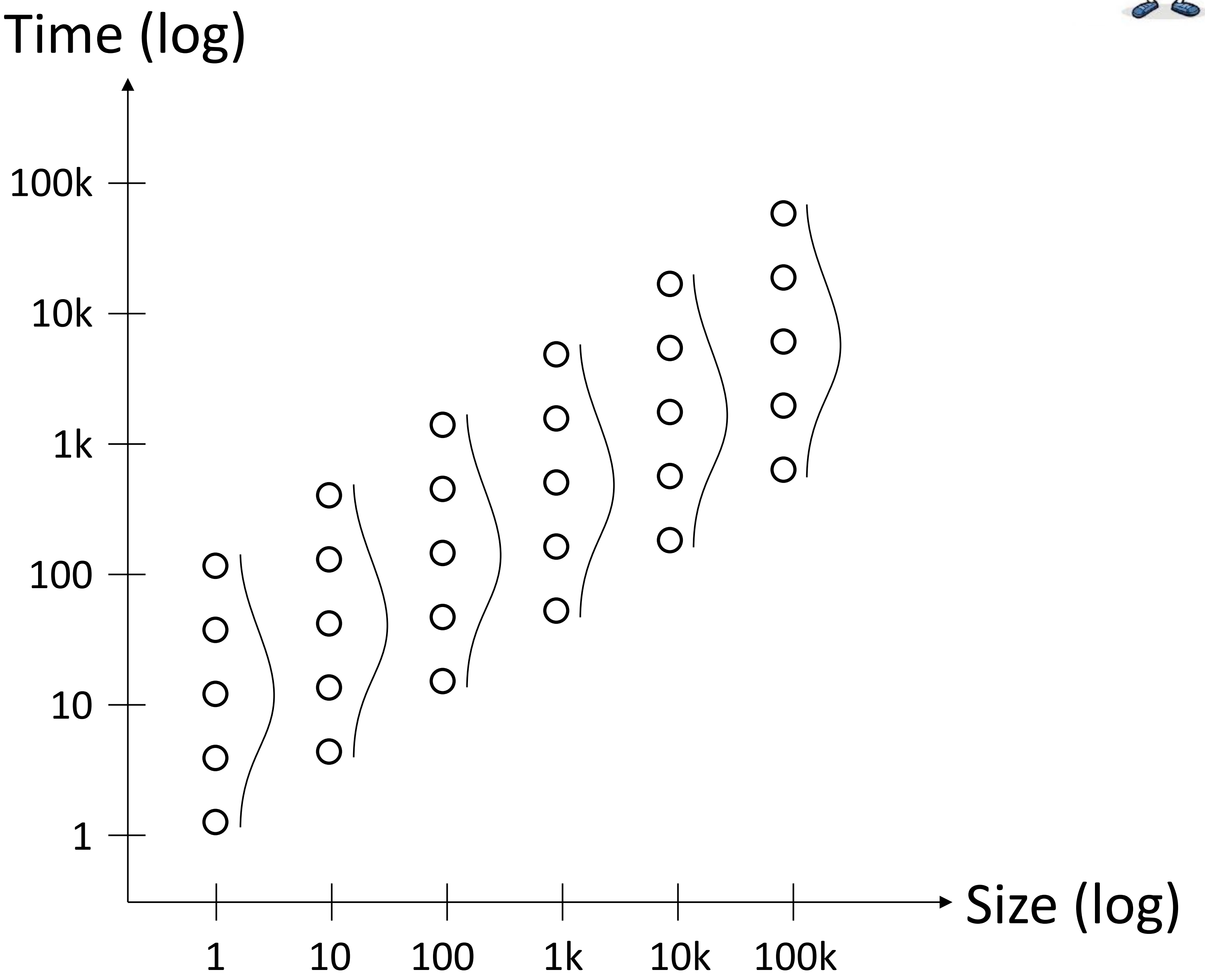
Possibly interesting class scribe: Why is Excel Turing-complete?

Source: https://www.microsoft.com/en-us/research/blog/lambda-the-ultimatae-excel-worksheet-function/

Revisiting our question from first class

Question: How to deal with cut-offs when binning?

- These are the true points that you would get if you could run the experiments long enough.
  - Notice the loglog scale

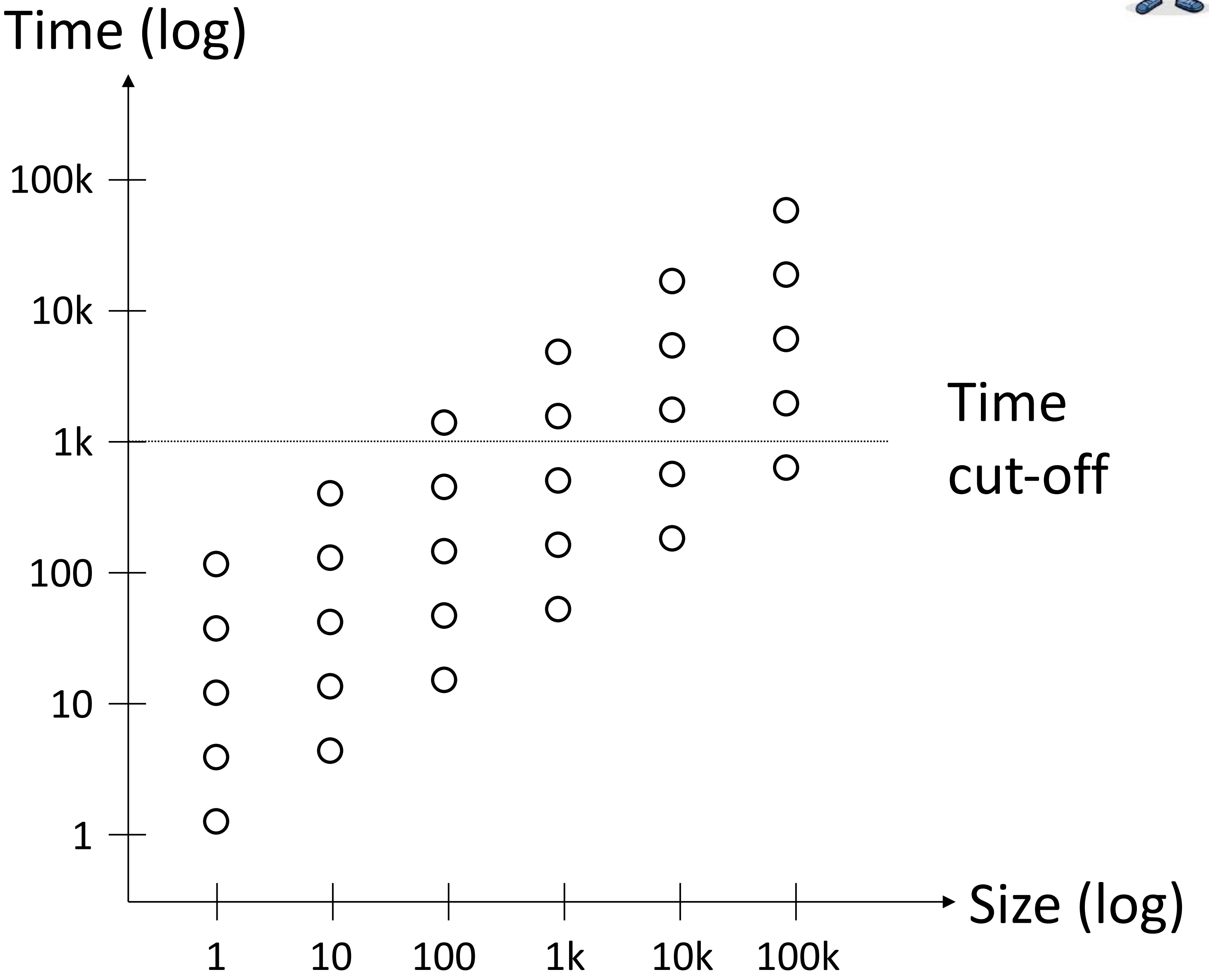


Question: How to deal with cut-offs when binning?

- These are the true points that you would get if you could run the experiments long enough.
  - Notice the loglog scale
- However, we can't and thus in practice cut-off the experiments after some time.

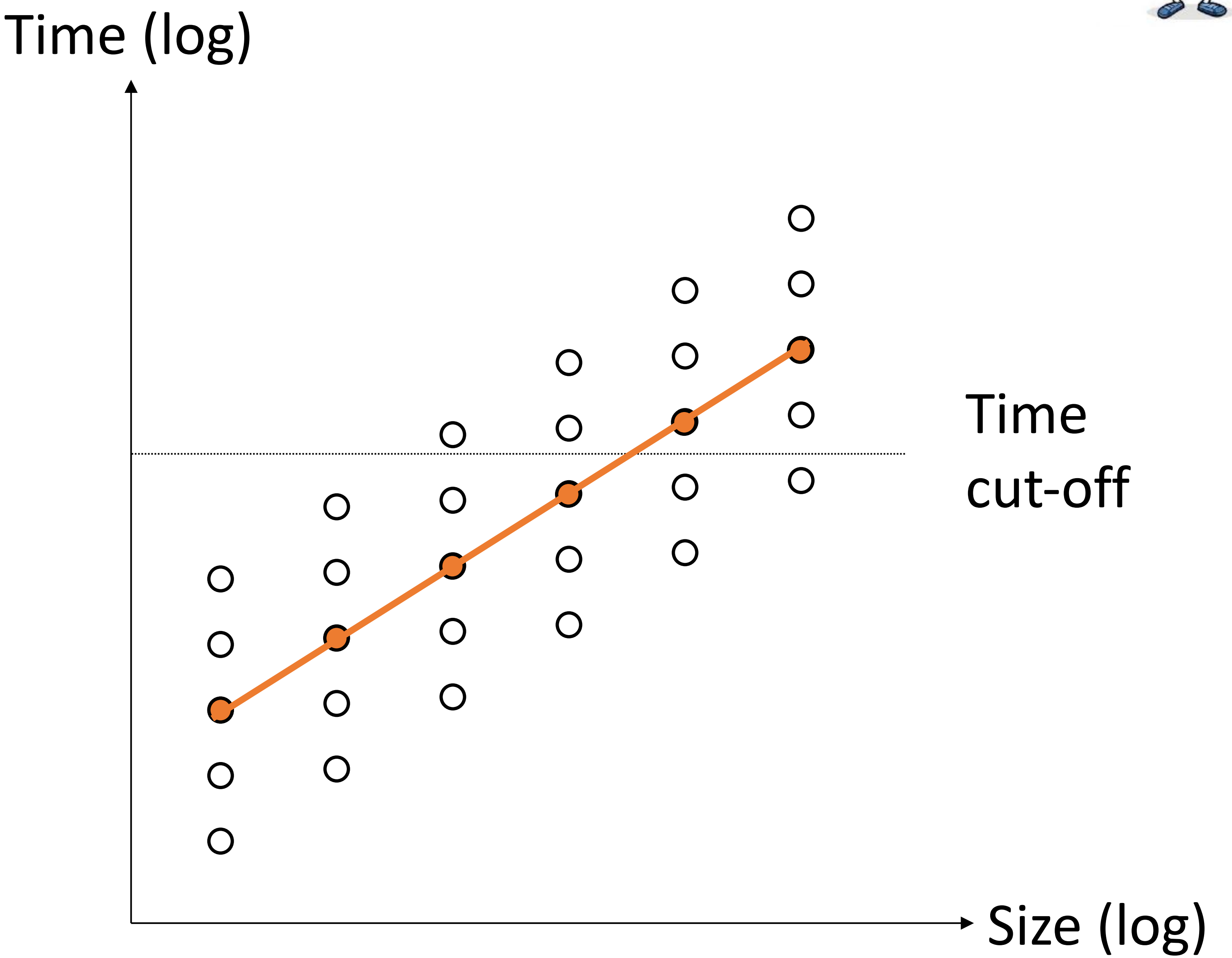
- Question: There is an overall trend, yet some variation for each experiment. We would still like to capture the trend with some smart aggregations. What can we do?

?



Question: How to deal with cut-offs when binning

- Here is what the aggregate would look like like if we could get all points and then aggregated for each size

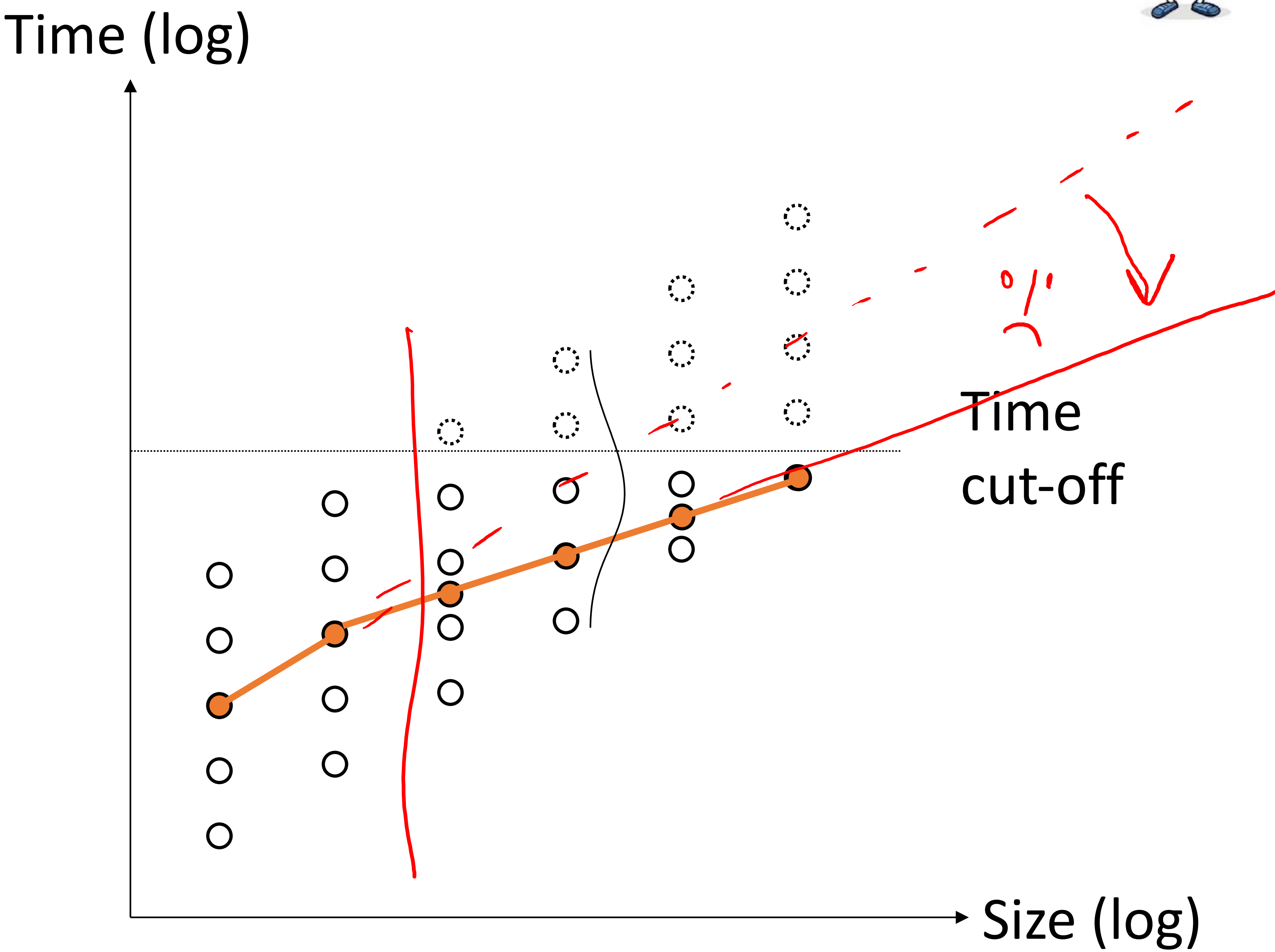


Question: How to deal with cut-offs when binning

- Here is what happens if we throw away all those points that take longer than the cut-off, and only average over the "seen points"

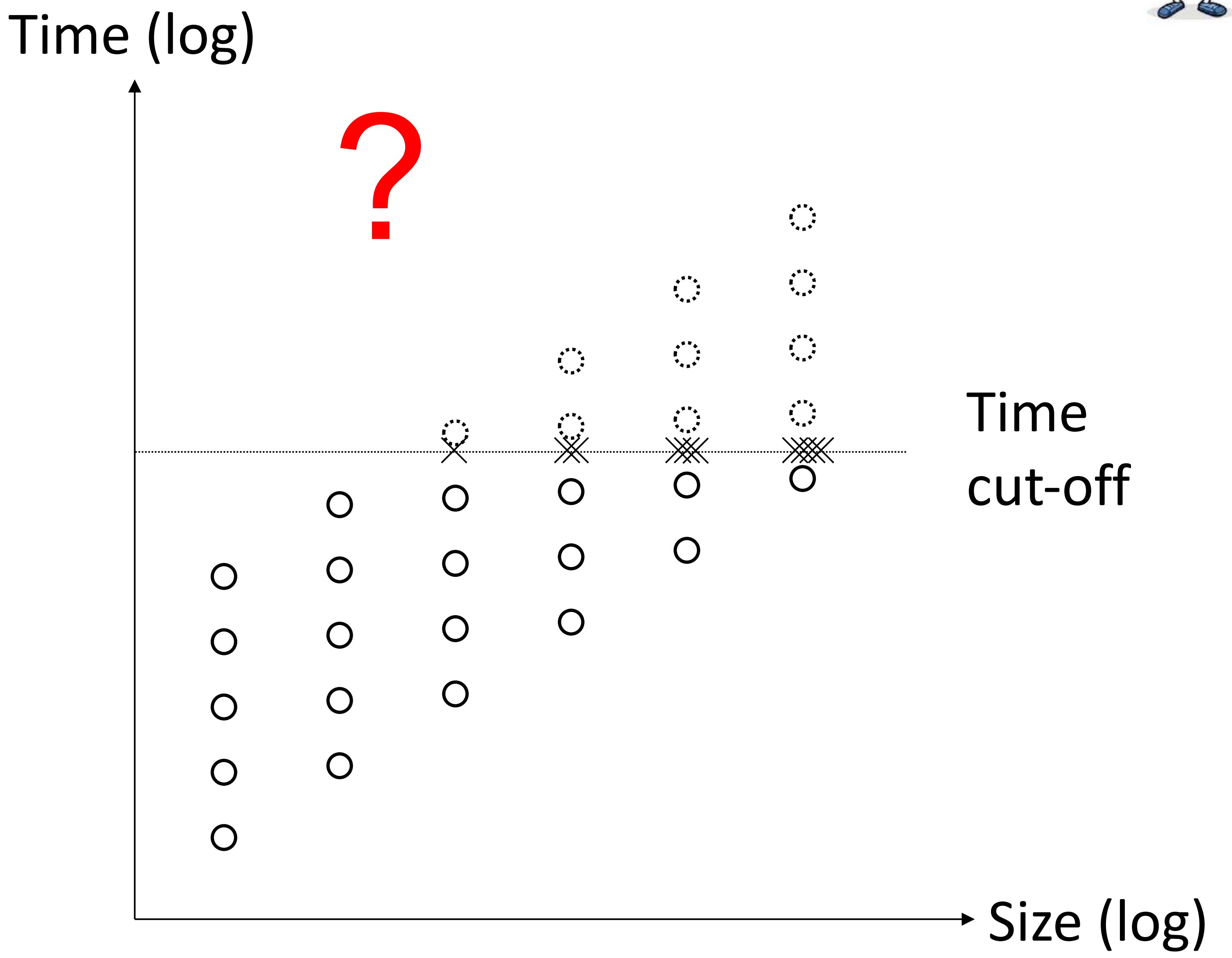
What would you do ?

We will discuss next class



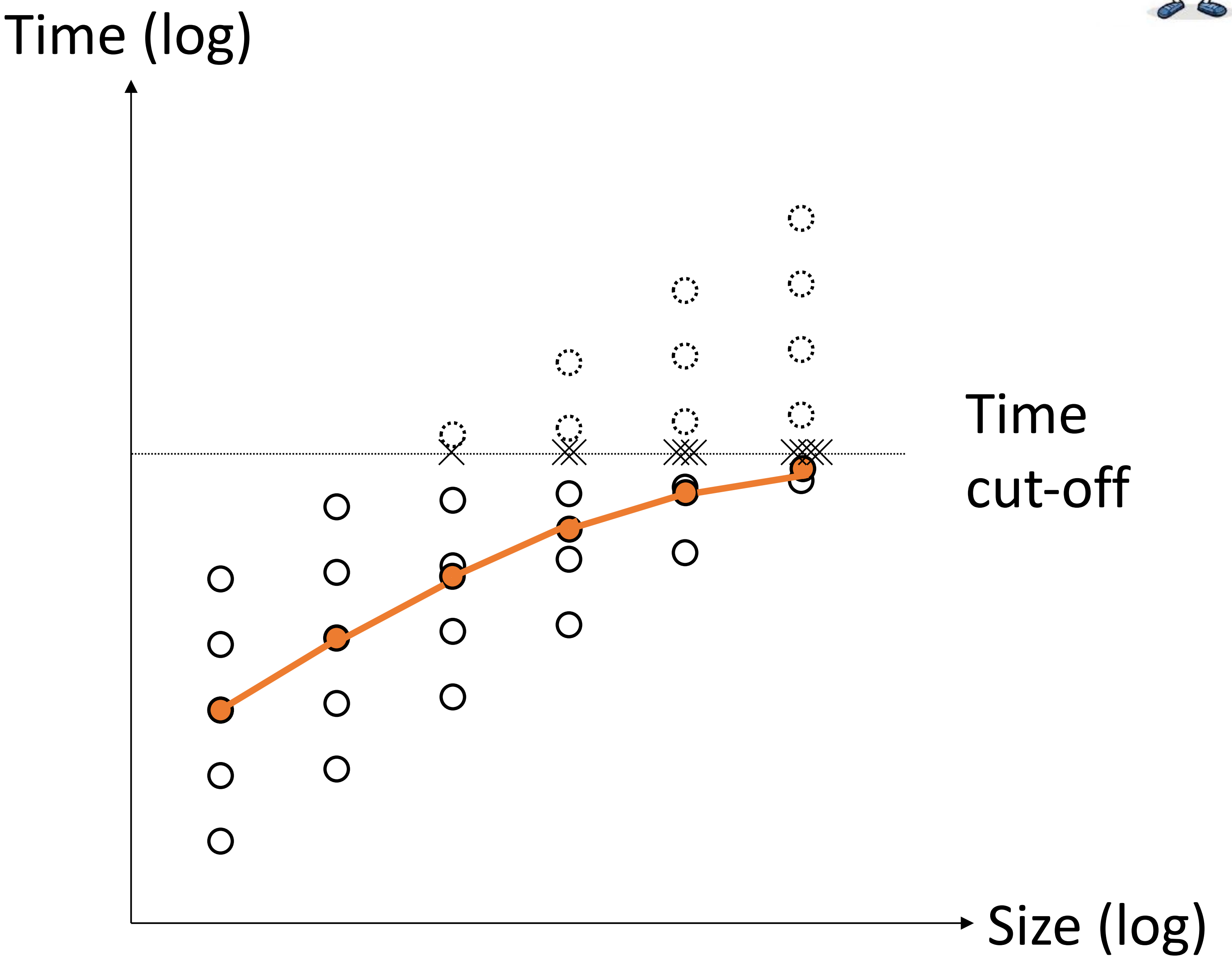
How to deal with cut-offs when binning: Option 1

- Option 1: Here is what happens if we cut the points off and still use the points, and then average



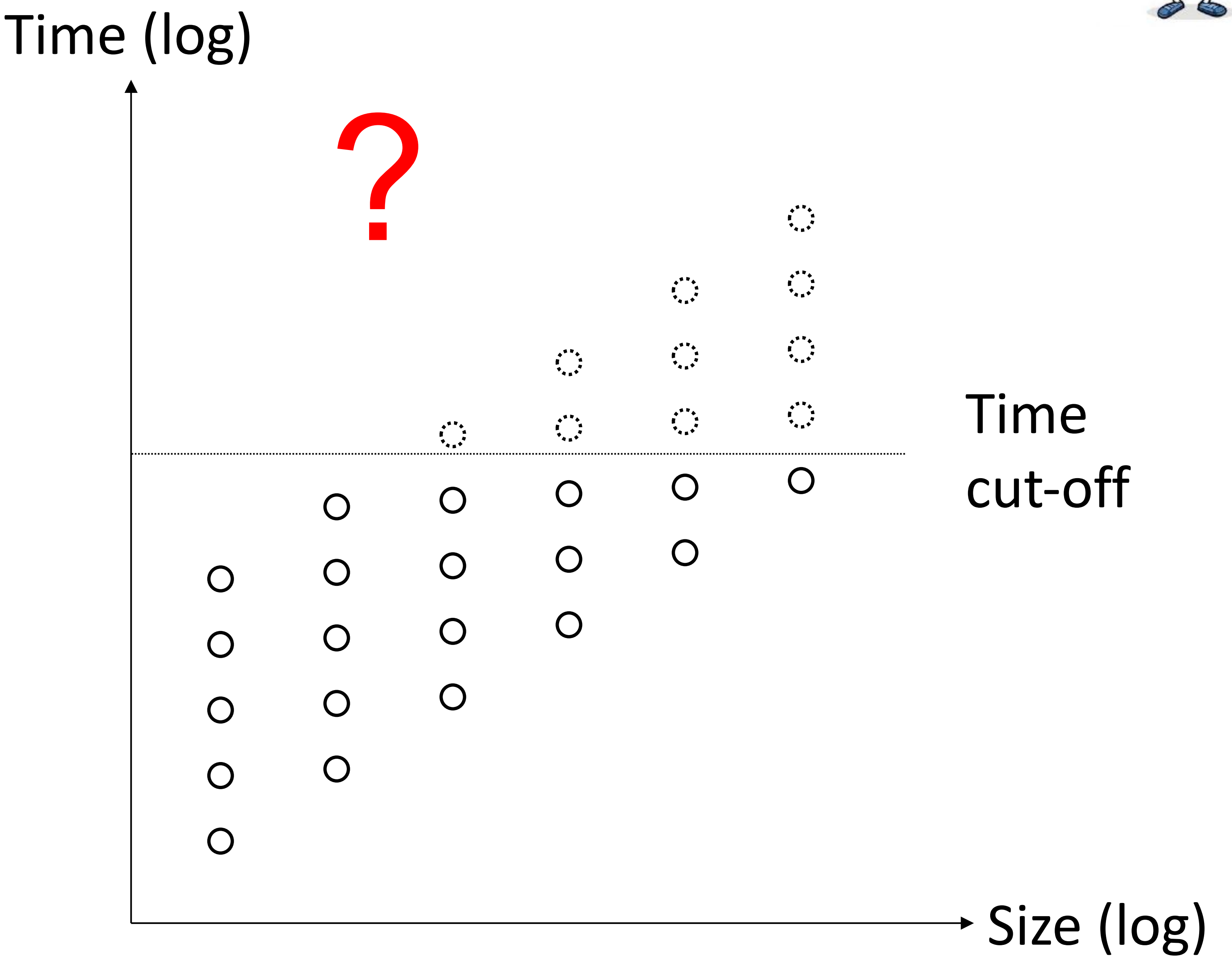
How to deal with cut-offs when binning: Option 1

- Option 1: Here is what happens if we cut the points off and still use the points, and then average



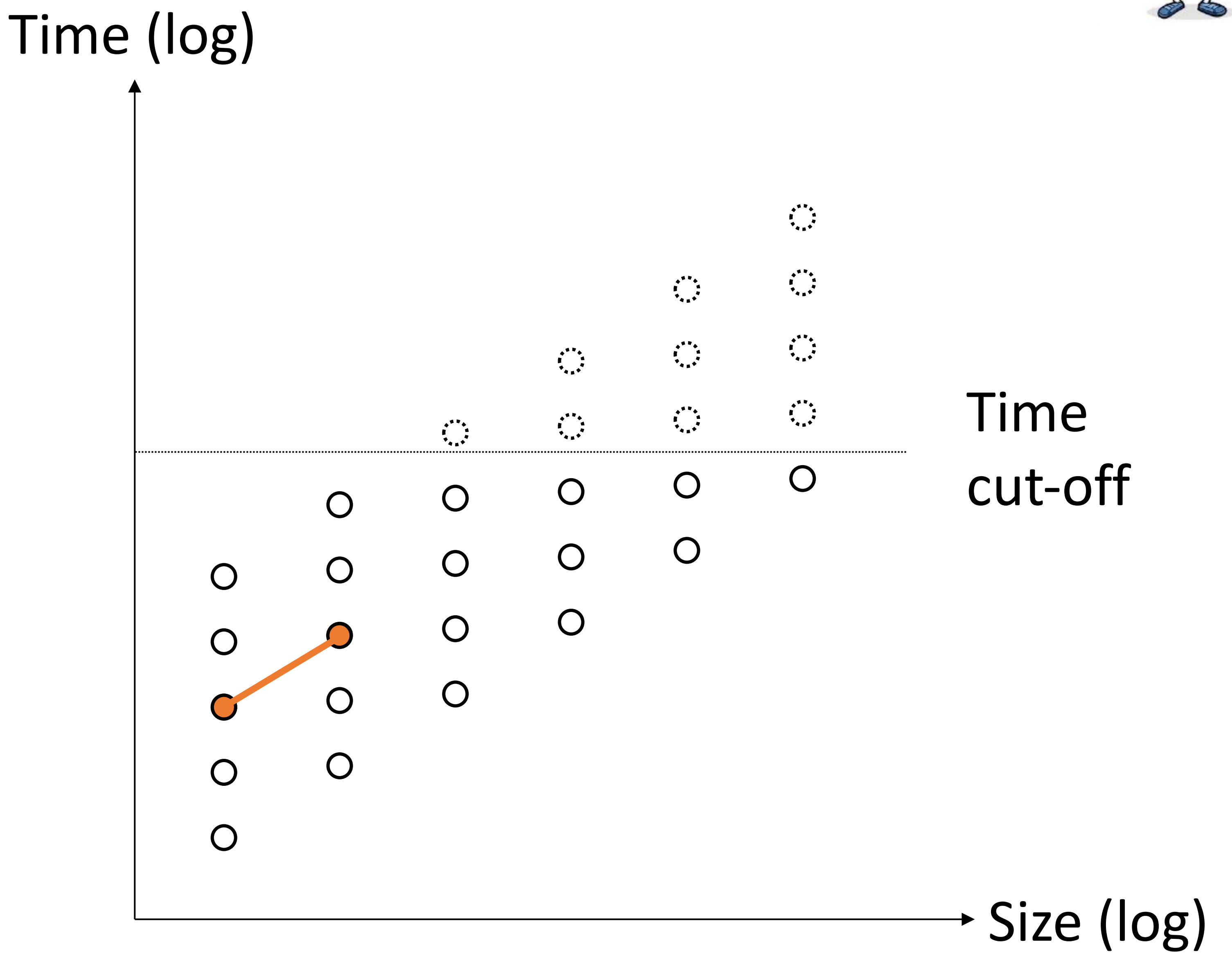
How to deal with cut-offs when binning: Option 2

- Option 1: Here is what happens if we cut the points off and still use the points, and then average
- Option 2: Here is what we can do if we *only* use those sizes for which all experiments finish in time



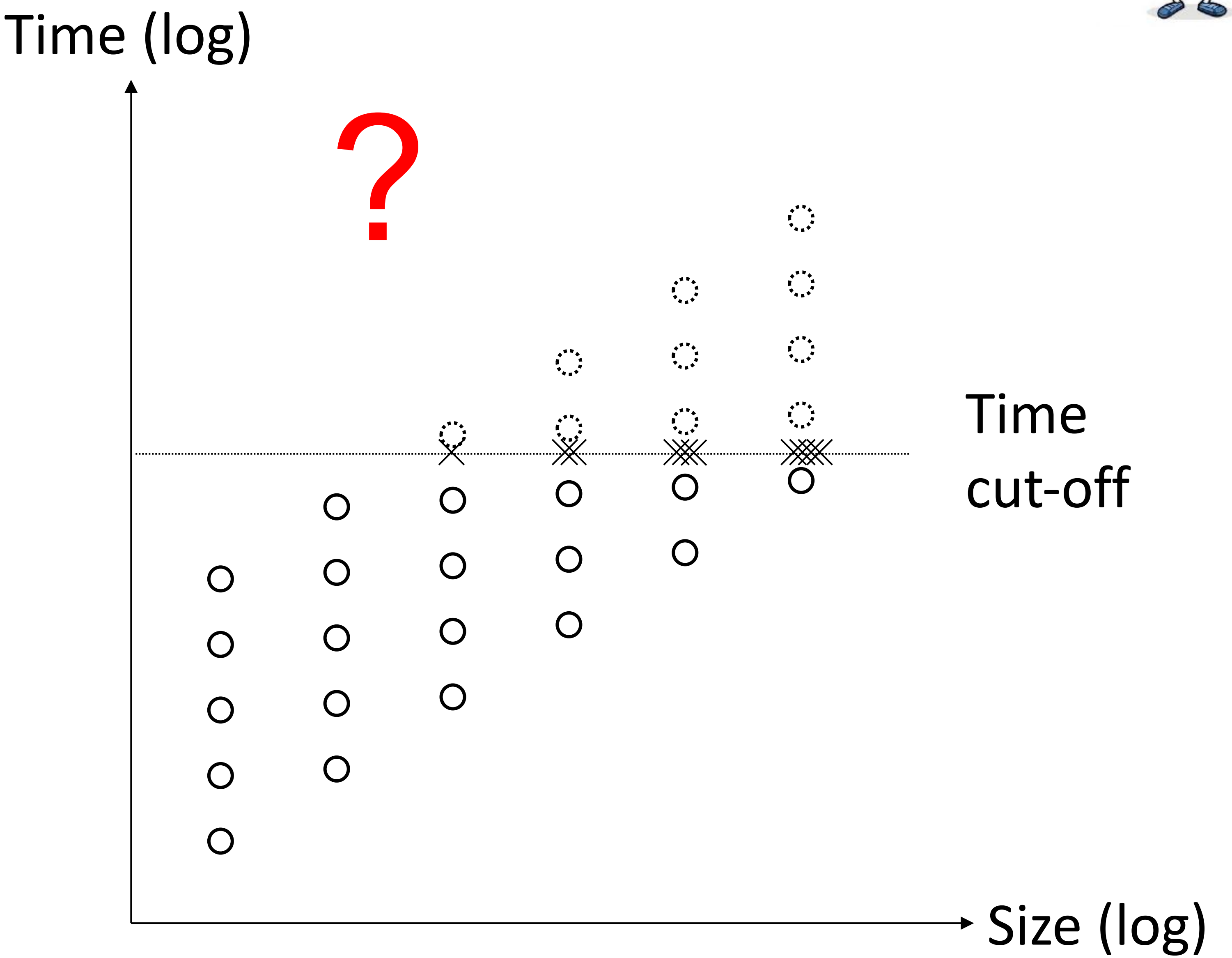
How to deal with cut-offs when binning: Option 2

- Option 1: Here is what happens if we cut the points off and still use the points, and then average
- Option 2: Here is what we can do if we *only* use those sizes for which all experiments finish in time



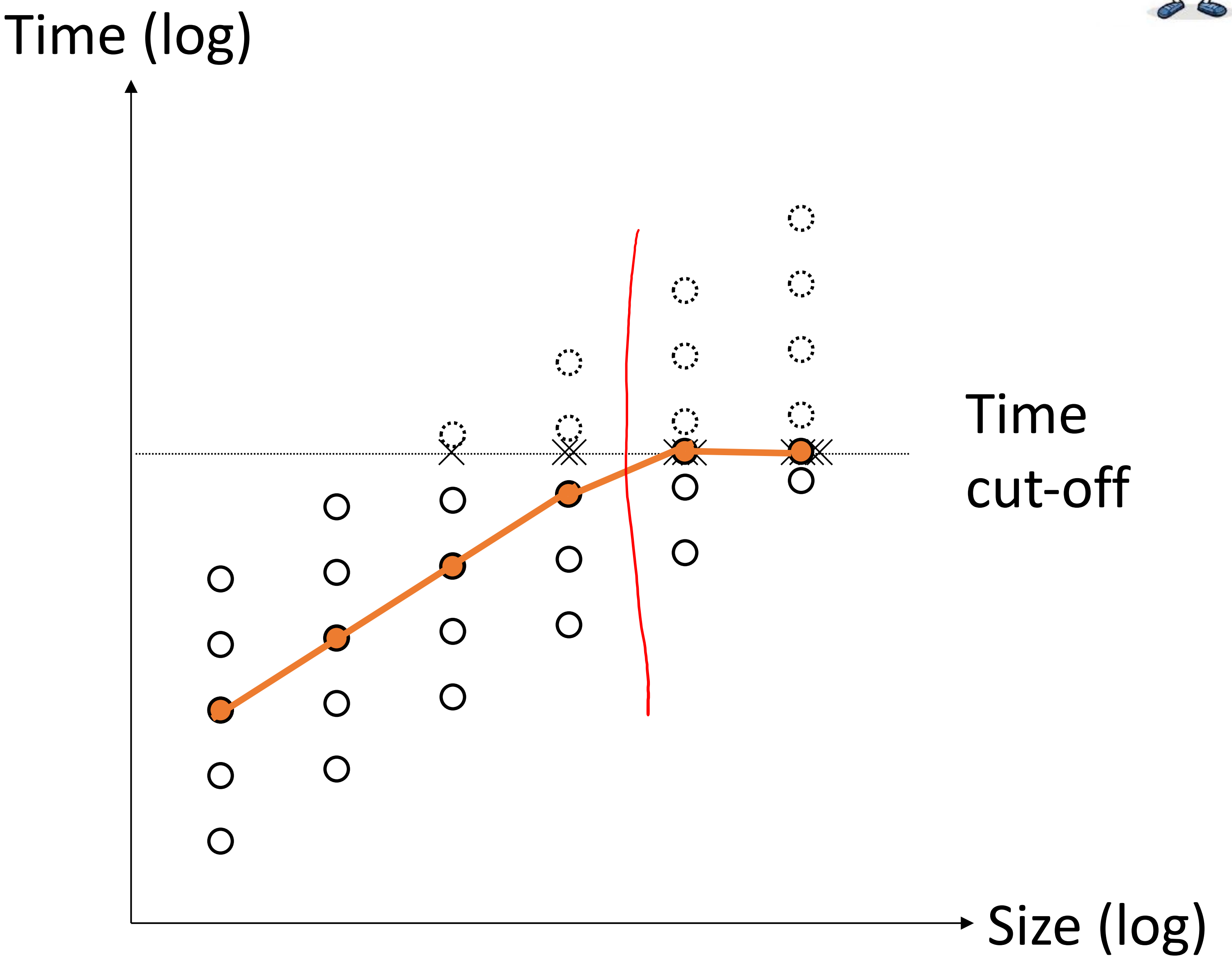
How to deal with cut-offs when binning: Option 3

- Option 1: Here is what happens if we cut the points off and still use the points, and then average
- Option 2: Here is what we can do if we *only* use those sizes for which all experiments finish in time
- Option 3: Here is what happens if we take the median over all seen and cut-off points



How to deal with cut-offs when binning: Option 3

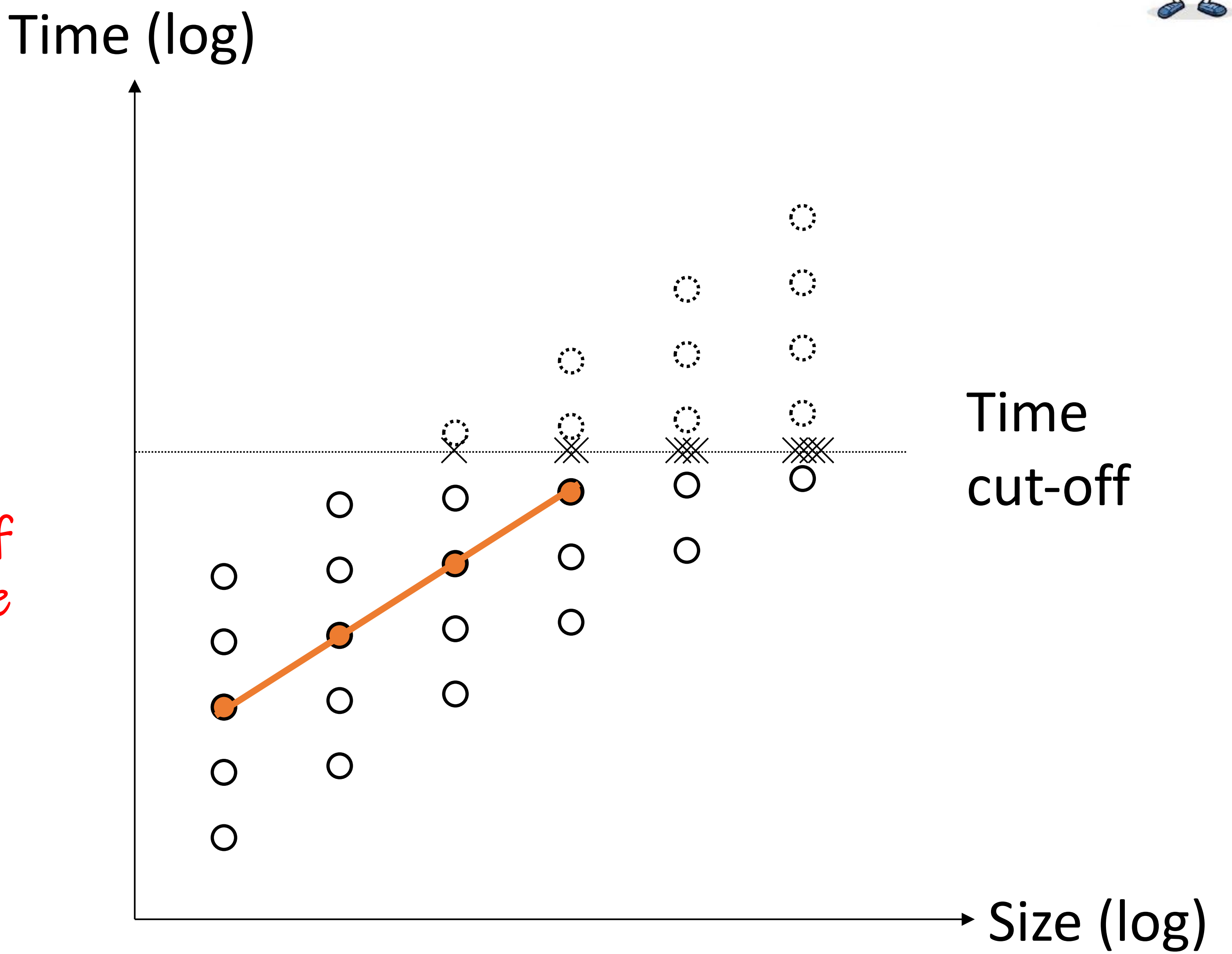
- Option 1: Here is what happens if we cut the points off and still use the points, and then average
- Option 2: Here is what we can do if we *only* use those sizes for which all experiments finish in time
- Option 3: Here is what happens if we take the median over all seen and cut-off points



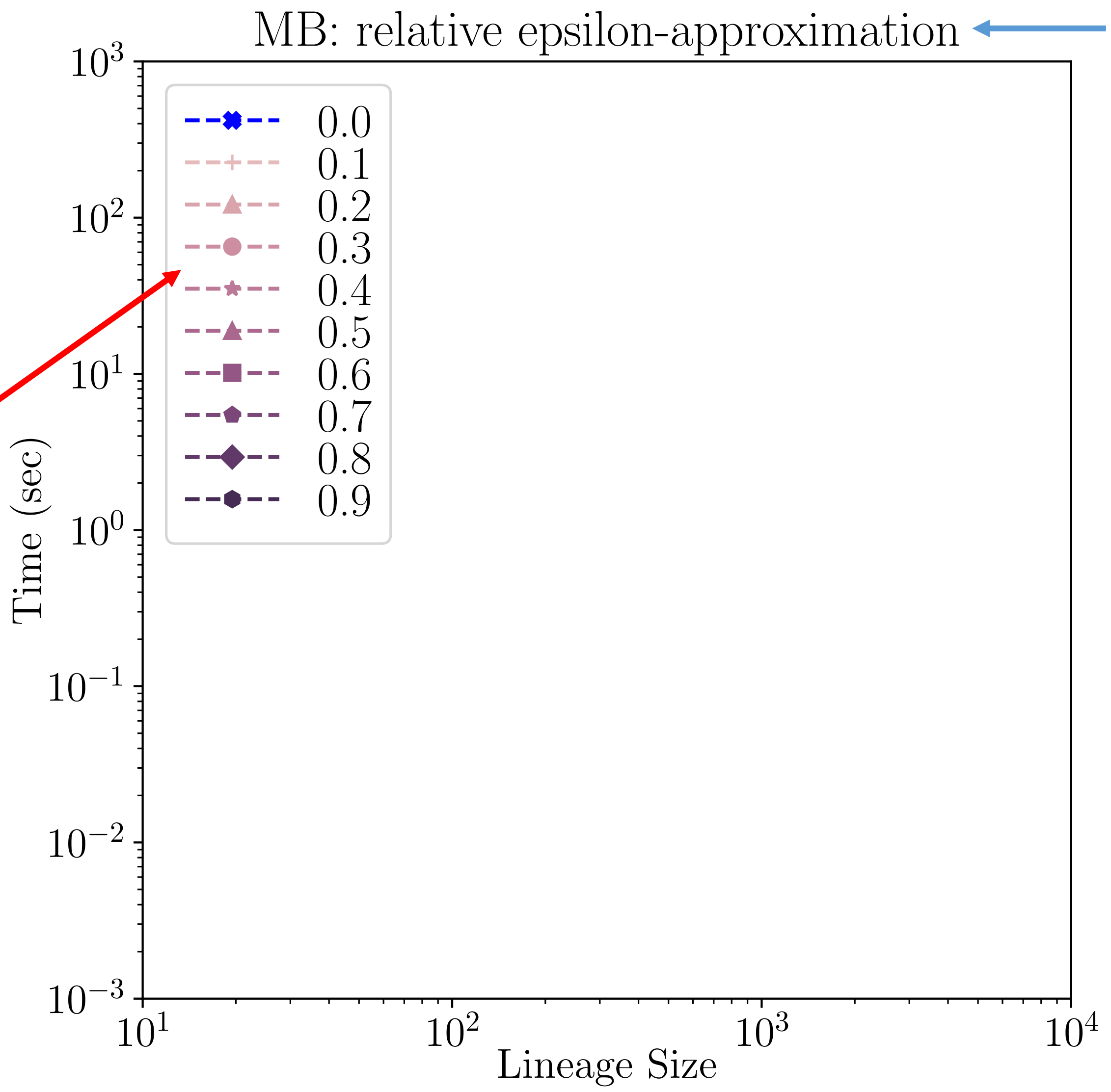
How to deal with cut-offs when binning: Suggestion

- Suggestion: Here is what happens if we take the median over all seen and cut-off points, as long as there are <50% cut-off points

Notice the informal "semantics" of median: If more points are "above you" then you are pulled by their number, not by their distance (in contrast to average where distance is kind of a weight)



Example: Experiments figures from Van der Heuvel+ [SIGMOD'19]



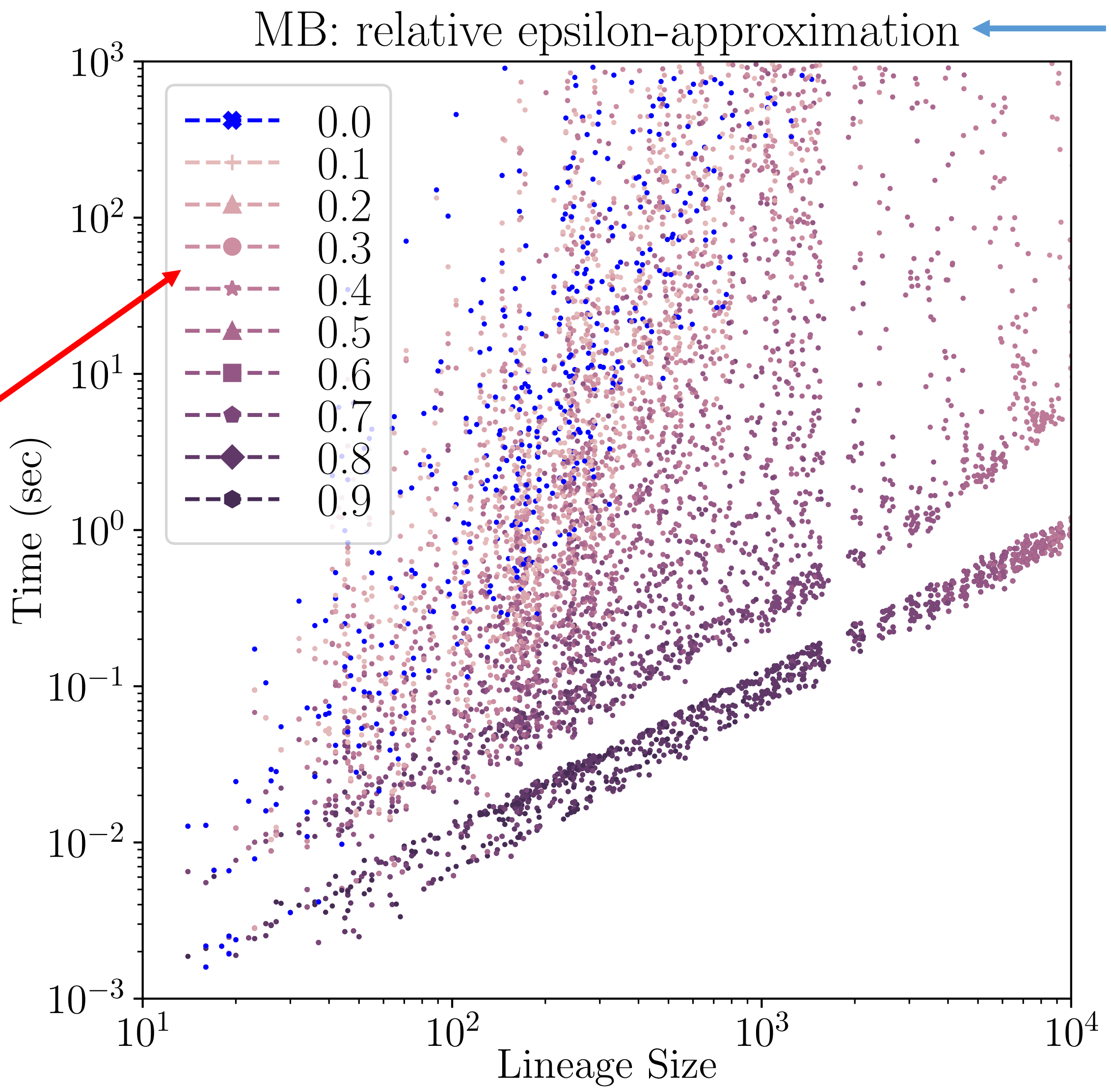
MB (prior): model-based
10 random bounds

Some type of error guarantees (smaller is better)

notice the log log scale!

Source: Van der Heuvel, Ivanov, Gatterbauer, Geerts, Theobald. Anytime approximation in probabilistic databases via scaled dissociations. SIGMOD 2019. https://doi.org/10.1145/3299869.3319900

Example: Experiments figures from Van der Heuvel+ [SIGMOD'19]



MB (prior): model-based
10 random bounds

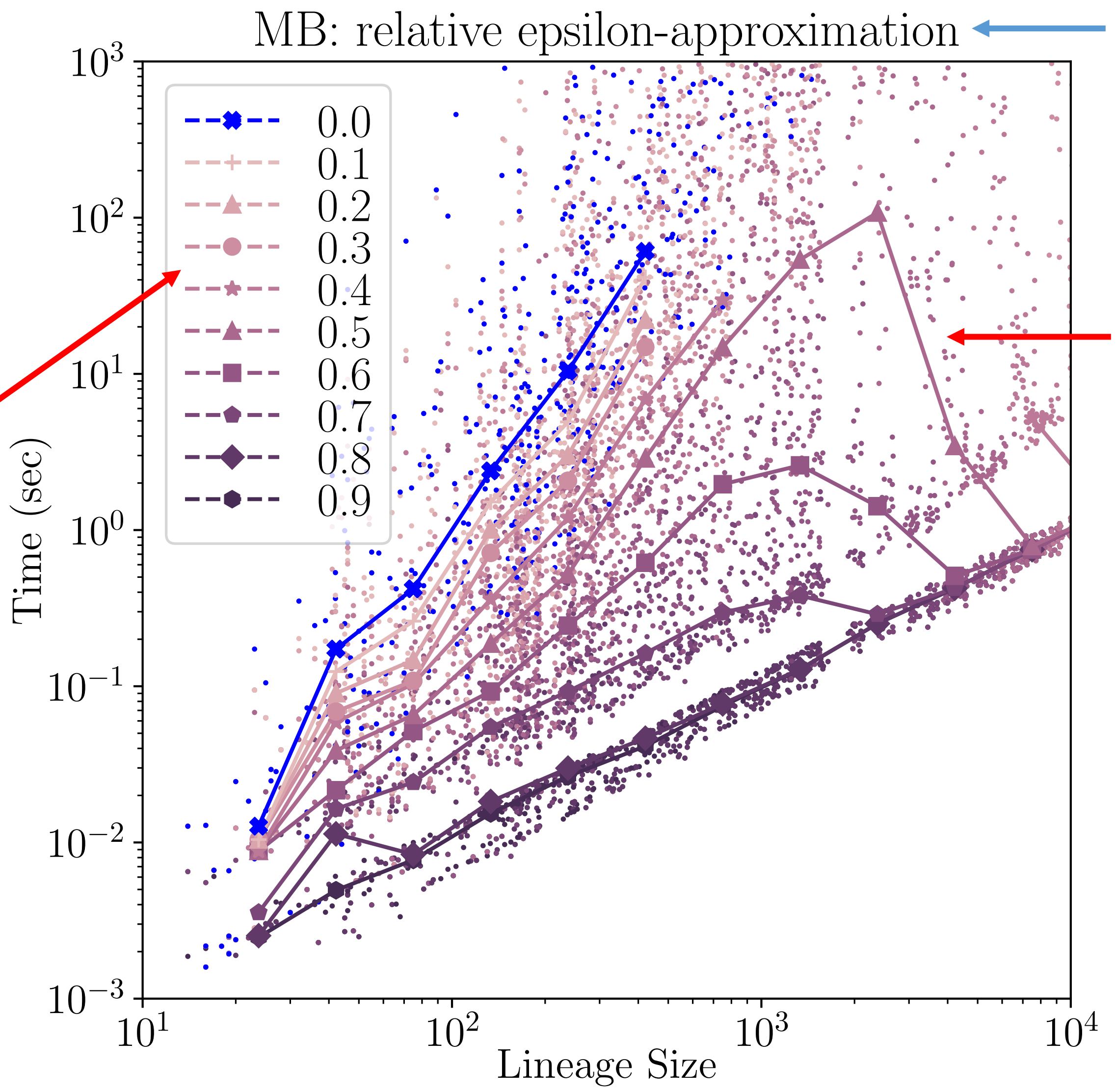
Some type of error guarantees (smaller is better)

notice the log log scale!

Source: Van der Heuvel, Ivanov, Gatterbauer, Geerts, Theobald. Anytime approximation in probabilistic databases via scaled dissociations. SIGMOD 2019. https://doi.org/10.1145/3299869.3319900

Example: Experiments figures from Van der Heuvel+ [SIGMOD'19]

MB (prior): model-based 10 random bounds



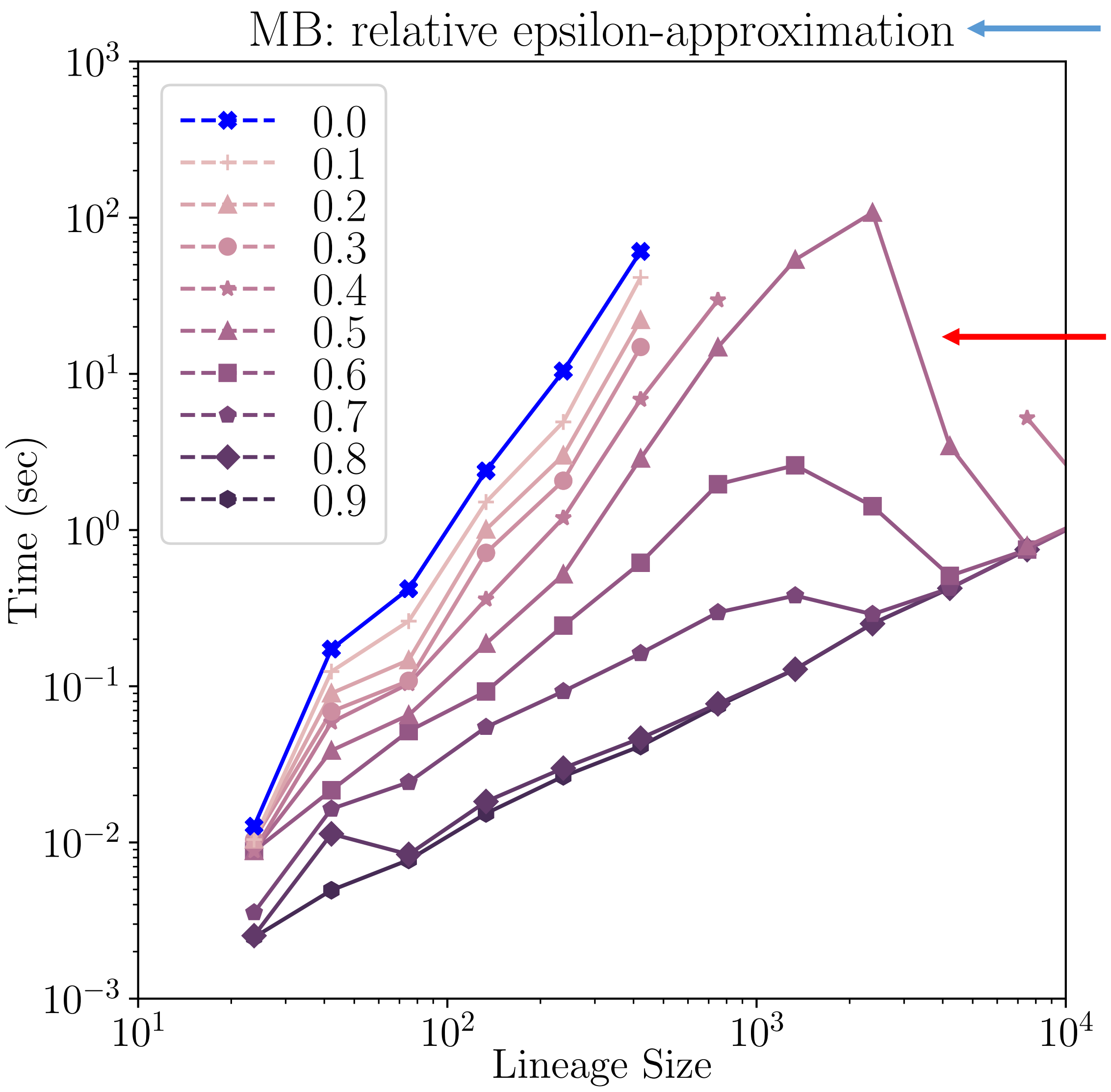
Median time to reach a certain error guarantee for fixed lin. size

Some type of error guarantees (smaller is better)

notice the log log scale!

Source: Van der Heuvel, Ivanov, Gatterbauer, Geerts, Theobald. Anytime approximation in probabilistic databases via scaled dissociations. SIGMOD 2019. https://doi.org/10.1145/3299869.3319900

Example: Experiments figures from Van der Heuvel+ [SIGMOD'19]

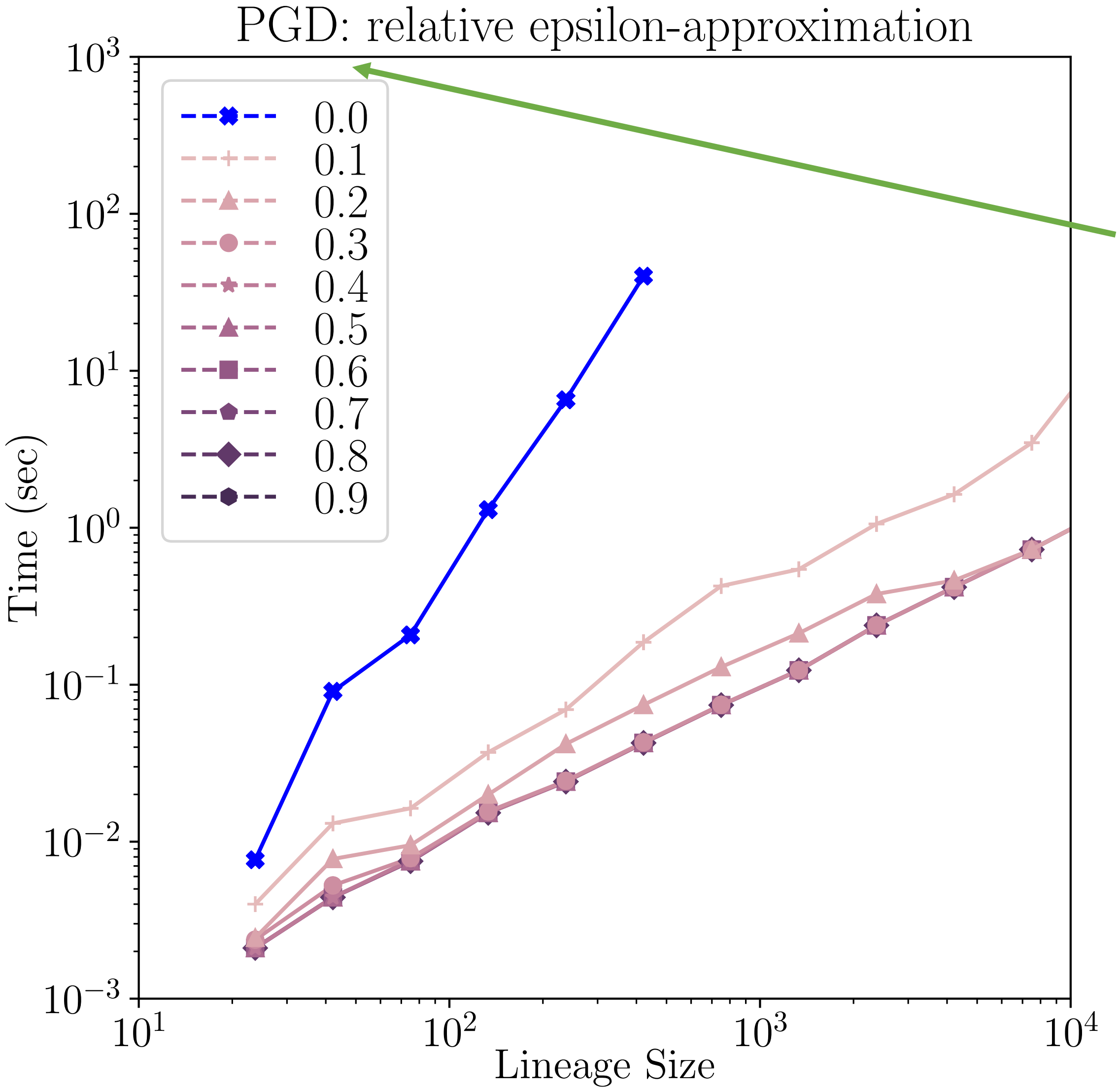


MB (prior): model-based
10 random bounds

Median time to reach a certain error guarantee for fixed lin. size

Source: Van der Heuvel, Ivanov, Gatterbauer, Geerts, Theobald. Anytime approximation in probabilistic databases via scaled dissociations. SIGMOD 2019. https://doi.org/10.1145/3299869.3319900

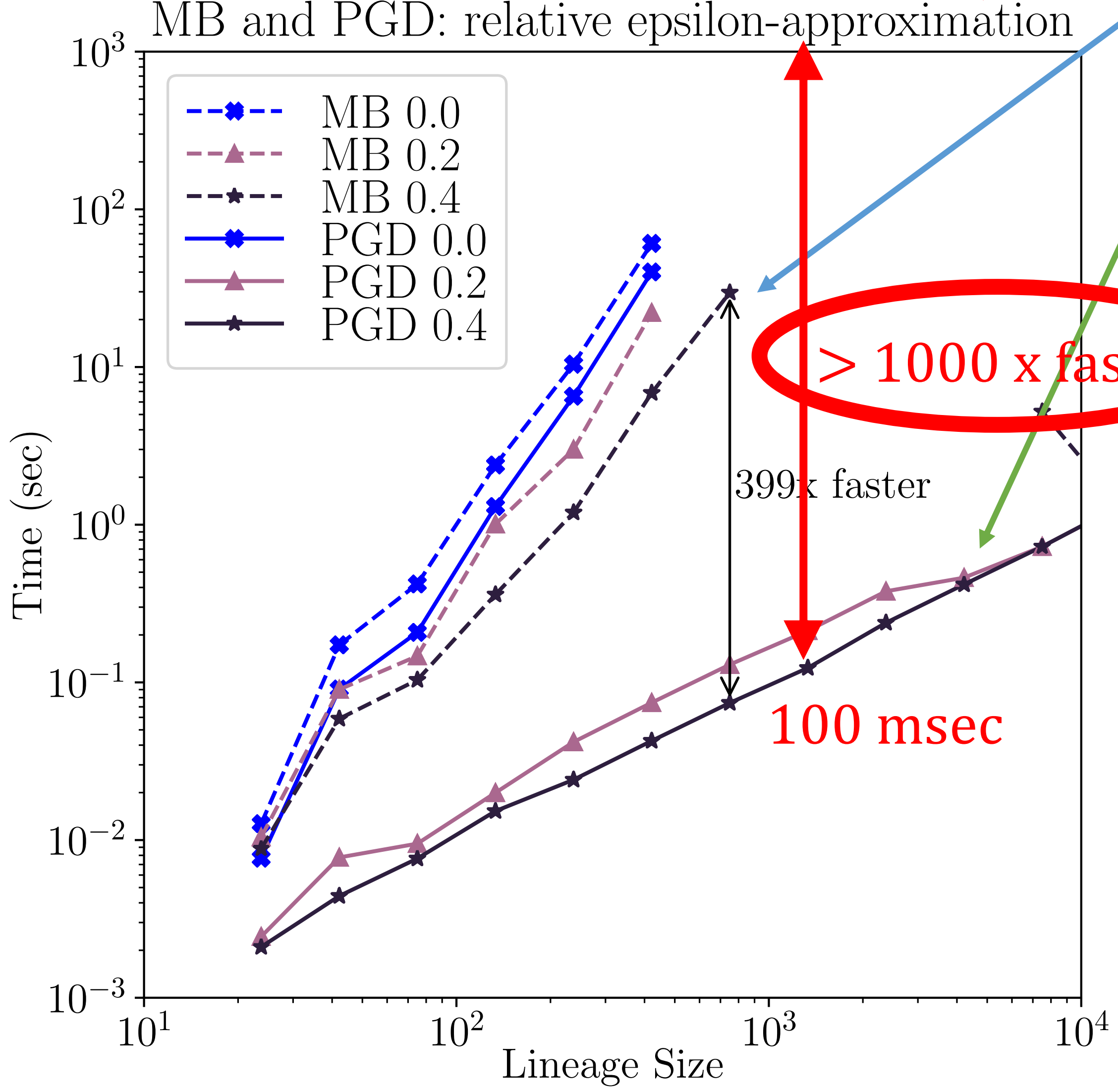
Example: Experiments figures from Van der Heuvel+ [SIGMOD'19]



PGD (our): projected gradient descent

Source: Van der Heuvel, Ivanov, Gatterbauer, Geerts, Theobald. Anytime approximation in probabilistic databases via scaled dissociations. SIGMOD 2019. https://doi.org/10.1145/3299869.3319900

Example: Experiments figures from Van der Heuvel+ [SIGMOD'19]



median >100 sec (timed out)

MB (prior): model-based
10 random bounds

PGD (our): projected
gradient descent

> 1000 x faster

100 msec

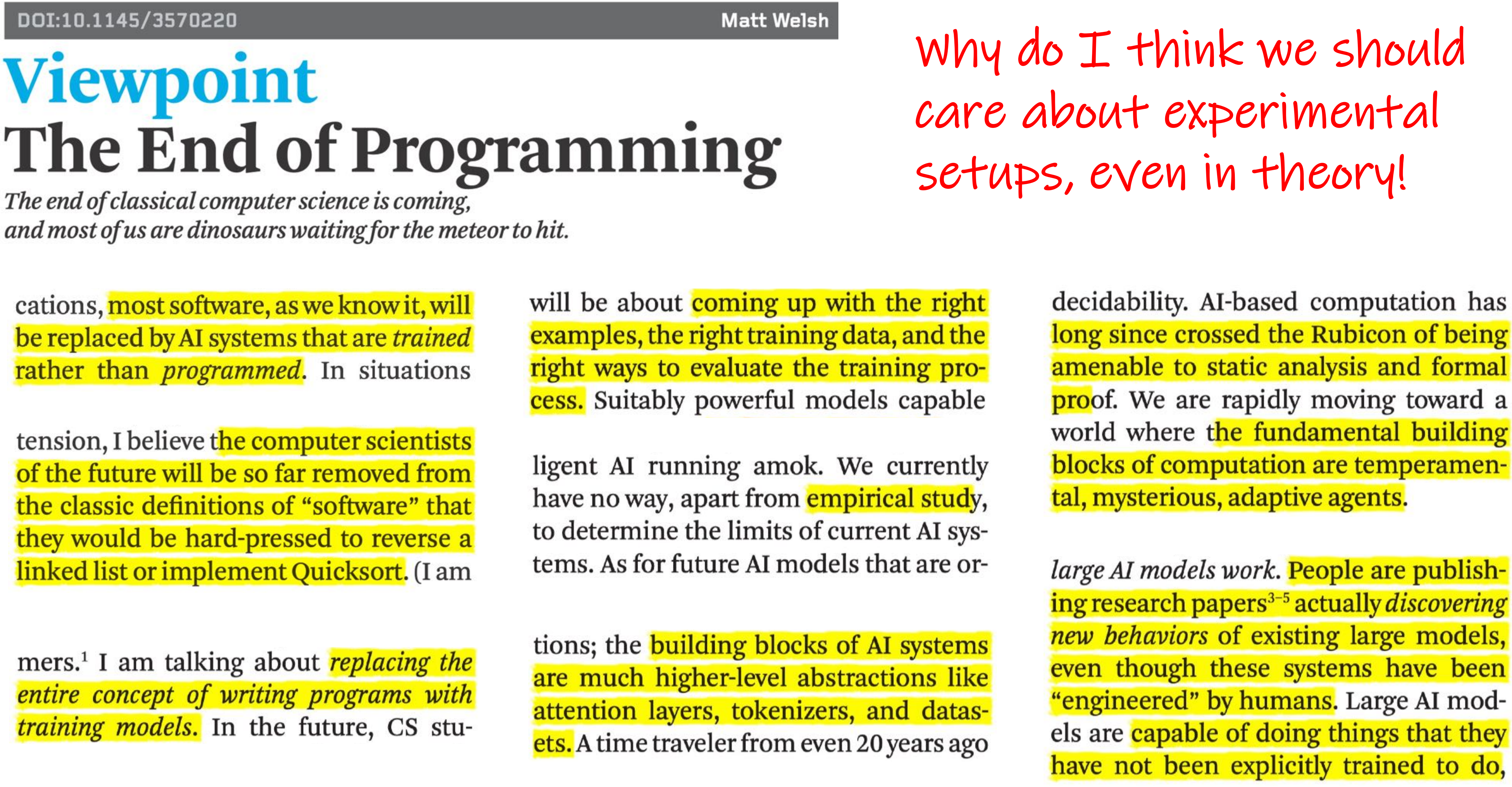
Take-away

- considerable speed-ups possible ☺

Source: Van der Heuvel, Ivanov, Gatterbauer, Geerts, Theobald. Anytime approximation in probabilistic databases via scaled dissociations. SIGMOD 2019. https://doi.org/10.1145/3299869.3319900

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Why do I think we should care about experimental setups, even in theory!



DOI:10.1145/3570220

Matt Welsh

Viewpoint

The End of Programming

The end of classical computer science is coming, and most of us are dinosaurs waiting for the meteor to hit.

cations, most software, as we know it, will be replaced by AI systems that are *trained* rather than *programmed*. In situations

tension, I believe the computer scientists of the future will be so far removed from the classic definitions of "software" that they would be hard-pressed to reverse a linked list or implement Quicksort. (I am

mers.[1] I am talking about *replacing the entire concept of writing programs with training models.* In the future, CS stu-

will be about coming up with the right examples, the right training data, and the right ways to evaluate the training process. Suitably powerful models capable

ligent AI running amok. We currently have no way, apart from empirical study, to determine the limits of current AI systems. As for future AI models that are or-

tions; the building blocks of AI systems are much higher-level abstractions like attention layers, tokenizers, and datasets. A time traveler from even 20 years ago

decidability. AI-based computation has long since crossed the Rubicon of being amenable to static analysis and formal proof. We are rapidly moving toward a world where the fundamental building blocks of computation are temperamental, mysterious, adaptive agents.

large AI models work. People are publishing research papers[3–5] actually *discovering new behaviors* of existing large models, even though these systems have been "engineered" by humans. Large AI models are capable of doing things that they have not been explicitly trained to do,

Source: Matt Welsh, CAMC 2023. https://cacm.acm.org/magazines/2023/1/267976-the-end-of-programming/fulltext, https://doi.org/10.1145/3570220

Outline: T1-U1: SQL

- SQL
    - Schema, keys, referential integrity
    - Joins
    - Aggregates and grouping
    - Nested queries (Subqueries)
    - Theta Joins
    - Nulls & Outer joins
    - Top-k
    - [Recursion: moved to T1-U4: Datalog]

Joins

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Q: Find all products under $200 manufactured in Japan;
return their names and prices!

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Joins

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Q: Find all products under $200 manufactured in Japan; return their names and prices!

```
SELECT  pName, price
FROM    Product, Company
WHERE   manufacturer=cName
   and  country='Japan'
   and  price <= 200
```

Join b/w Product and Company

PName	Price
SingleTouch	$149.99

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Joins

302

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT  *
FROM    Product, Company
WHERE   manufacturer=cName
```

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Joins

302

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

PName	Price	Category	Manufacturer	CName	StockPrice	Country
Gizmo	$19.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
Powergizmo	$29.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
SingleTouch	$149.99	Photography	Canon	Canon	65	Japan
MultiTouch	$203.99	Household	Hitachi	Hitachi	15	Japan

```
SELECT  *
FROM    Product, Company
WHERE   manufacturer=cName
```

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

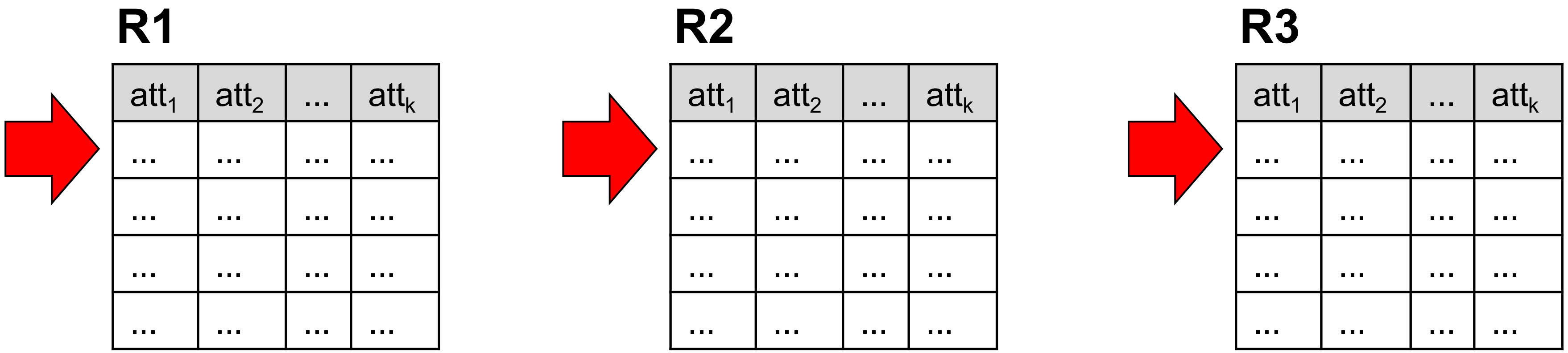
Meaning (Semantics) of conjunctive SQL Queries

```
SELECT  a_1, a_2, …, a_k
FROM    R_1 as x_1, R_2 as x_2, …, R_n as x_n
WHERE   Conditions
```

Conceptual evaluation strategy (nested for loops):

```
Answer = {}
for $x_1$ in $R_1$ do
    for $x_2$ in $R_2$ do
        …..
            for $x_n$ in $R_n$ do
                if Conditions
                    then Answer = Answer $\cup$ {($a_1$,…,$a_k$)}
return Answer
```

Meaning (Semantics) of conjunctive SQL Queries



R1

att_1	att_2	…	att_k
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…

R2

att_1	att_2	…	att_k
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…

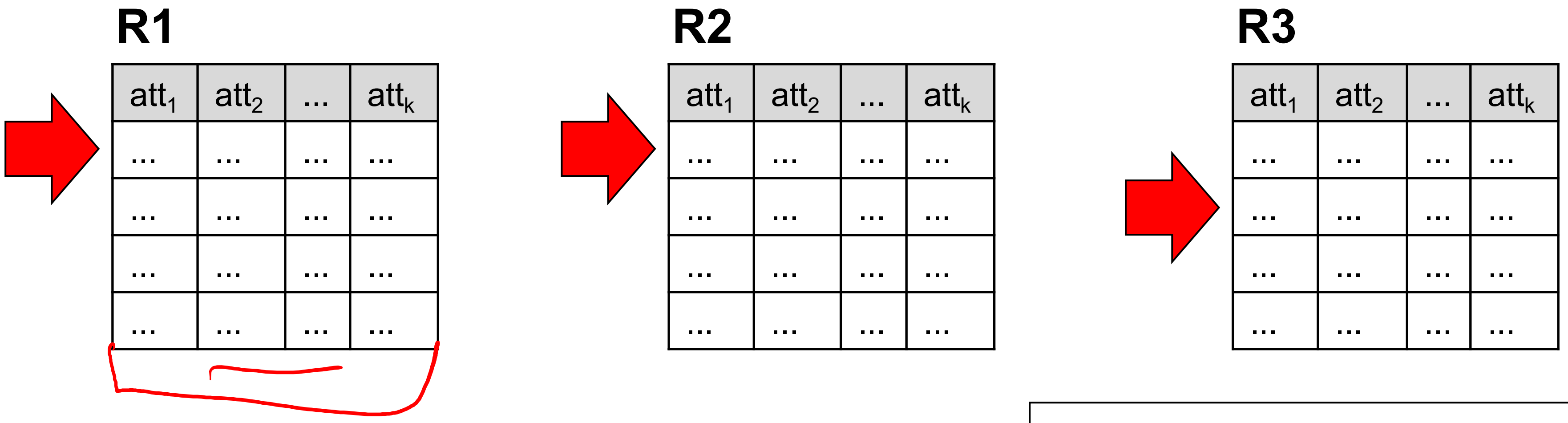
R3

att_1	att_2	…	att_k
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…

Conceptual evaluation strategy (nested for loops):

```
Answer = {}
for $x_1$ in $R_1$ do
    for $x_2$ in $R_2$ do
        .....
            for $x_n$ in $R_n$ do
                if Conditions
                    then Answer = Answer $\cup$ {($a_1$,…,$a_k$)}
return Answer
```

Meaning (Semantics) of conjunctive SQL Queries



R1

att_1	att_2	…	att_k
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…

R2

att_1	att_2	…	att_k
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…

R3

att_1	att_2	…	att_k
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…

```
Answer = {}
for $x_1$ in $R_1$ do
    for $x_2$ in $R_2$ do
        .....
            for $x_n$ in $R_n$ do
                if Conditions
                    then Answer = Answer ∪ {($a_1$,…,$a_k$)}
return Answer
```

Notice that those queries are "monotone": whenever we add tuples to the input, the output can never decrease:

if $R_1 \subseteq R_1', R_2 \subseteq R_2', R_3 \subseteq R_3'$

then $Q(R_1, R_2, R_3) \subseteq Q(R_1', R_2', R_3')$

Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - **FROM**: Compute the cross-product of relation-list.
  - **WHERE**: Discard resulting tuples if they fail qualifications ("select" the rest)
  - **SELECT**: Delete attributes that are not in target-list.
  - If **DISTINCT** is specified, eliminate duplicate rows.
- This strategy is probably the least efficient way to compute a query! An optimizer will find (algebraically equivalent but) more efficient strategies to compute the same answers.
- We say "semantics" not "execution order". Why?

?

Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - **FROM**: Compute the cross-product of relation-list.
  - **WHERE**: Discard resulting tuples if they fail qualifications ("select" the rest)
  - **SELECT**: Delete attributes that are not in target-list.
  - If **DISTINCT** is specified, eliminate duplicate rows.
- This strategy is probably the least efficient way to compute a query! An optimizer will find (algebraically equivalent but) more efficient strategies to compute the same answers.
- We say “semantics” not “execution order”. Why?
  - The preceding slides show what a join means (semantics = meaning): "the logic"
  - Not actually how the DBMS actually executes it (separation of concerns): algebra

Table Alias (Tuple Variables)

Person (pName, address, works_for)
University (uName, address)

```
SELECT  DISTINCT pName, address
FROM    Person, University
WHERE   works_for = uName
```

What will this query return ?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Table Alias (Tuple Variables)

Person (pName, address, works_for)
University (uName, address)

```
SELECT  DISTINCT pName, address
FROM    Person, University
WHERE   works_for = uName
```

which address?
Error!

```
SELECT  DISTINCT pName, University.address
FROM    Person, University
WHERE   Person.works_for = University.uName
```

```
SELECT  DISTINCT X.pName, Y.address
FROM    Person as X, University Y
WHERE   X.works_for = Y.uName
```

Notice that the use of "as" is not necessary, it is optional !!

Using the Formal Semantics

R(a), S(a), T(a)

What do these queries compute?

R

a
1
2

S

a
1

T

a
2

```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```

?

```
SELECT  R.a
FROM    R, S, T
WHERE   R.a=S.a
     or R.a=T.a
```

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Example originally proposed in Garcia-Molina, Ullman, Widom. Database Systems. 2001. Ch. 6.2.4 Interpreting Multirelation Queries. http://infolab.stanford.edu/~ullman/dscb.html

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Using the Formal Semantics

305

R(a), S(a), T(a)

R

a
1
2

S

a
1

T

a
2

What do these queries compute?

```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```

⇨

a
1

Returns R ∩ S
(intersection)

```
SELECT  R.a
FROM    R, S, T
WHERE   R.a=S.a
     or R.a=T.a
```

⇨ ?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Example originally proposed in Garcia-Molina, Ullman, Widom. Database Systems. 2001. Ch. 6.2.4 Interpreting Multirelation Queries. http://infolab.stanford.edu/~ullman/dscb.html

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Using the Formal Semantics

R(a), S(a), T(a)

What do these queries compute?

R

a
1
2

S

a
1

T

a
2

```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```

a
1

Returns R ∩ S
(intersection)

```
SELECT  R.a
FROM    R, S, T
WHERE   R.a=S.a
     or R.a=T.a
```

a
1
2

Returns R ∩ (S ∪ T)
if S ≠ ∅ and T ≠ ∅

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Example originally proposed in Garcia-Molina, Ullman, Widom. Database Systems. 2001. Ch. 6.2.4 Interpreting Multirelation Queries. http://infolab.stanford.edu/~ullman/dscb.html

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Using the Formal Semantics

R(a), S(a), T2(a)

305

R

a
1
2

S

a
1

T2

a
~~2~~

Our colorful hands represent "team exercises" If we are online, please make a screenshot!

Next, we are removing the input tuple "(2)"

What do these queries compute?

```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```

a
1

Returns R ∩ S (intersection)

?

```
SELECT  R.a
FROM    R, S, T2 as T
WHERE   R.a=S.a
    or  R.a=T.a
```

a
1
2

Returns R ∩ (S ∪ T)

if S ≠ ∅ and T ≠ ∅

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Example originally proposed in Garcia-Molina, Ullman, Widom. Database Systems. 2001. Ch. 6.2.4 Interpreting Multirelation Queries. http://infolab.stanford.edu/~ullman/dscb.html

Using the Formal Semantics

R(a), S(a), T2(a)

R

a
1
2

S

a
1

T2

a
~~2~~

Next, we are removing the input tuple "(2)"

What do these queries compute?

```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```

⇨

a
1

Returns R ∩ S (intersection)

```
SELECT  R.a
FROM    R, S, T2 as T
WHERE   R.a=S.a
    or  R.a=T.a
```

⇨

a
1
2

Returns R ∩ (S ∪ T) if S ≠ ∅ and T ≠ ∅

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Example originally proposed in Garcia-Molina, Ullman, Widom. Database Systems. 2001. Ch. 6.2.4 Interpreting Multirelation Queries. http://infolab.stanford.edu/~ullman/dscb.html

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Using the Formal Semantics

R(a), S(a), T2(a)

What do these queries compute?

R

a
1
2

S

a
1

T2

a
~~2~~

Next, we are removing the input tuple "(2)"

```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```

a
1

Returns R ∩ S (intersection)

```
SELECT  R.a
FROM    R, S, T2 as T
WHERE   R.a=S.a
     or R.a=T.a
```

a

Returns ∅ if S = ∅ or T = ∅

Can seem counterintuitive! But remember conceptual evaluation strategy: Nested loops. If one table is empty -> no looping

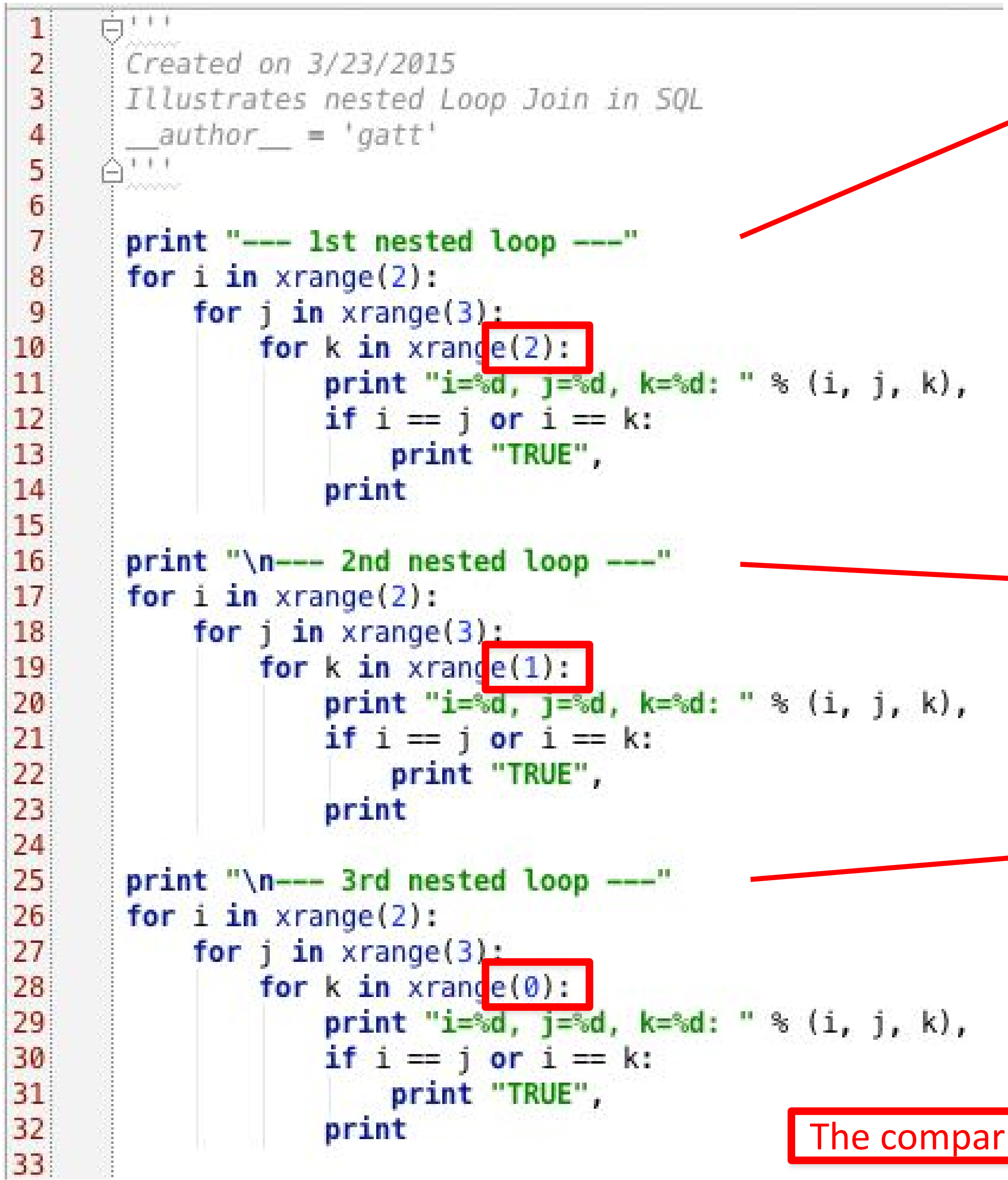
SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Example originally proposed in Garcia-Molina, Ullman, Widom. Database Systems. 2001. Ch. 6.2.4 Interpreting Multirelation Queries. http://infolab.stanford.edu/~ullman/dscb.html

Illustration with Python

306

Python file



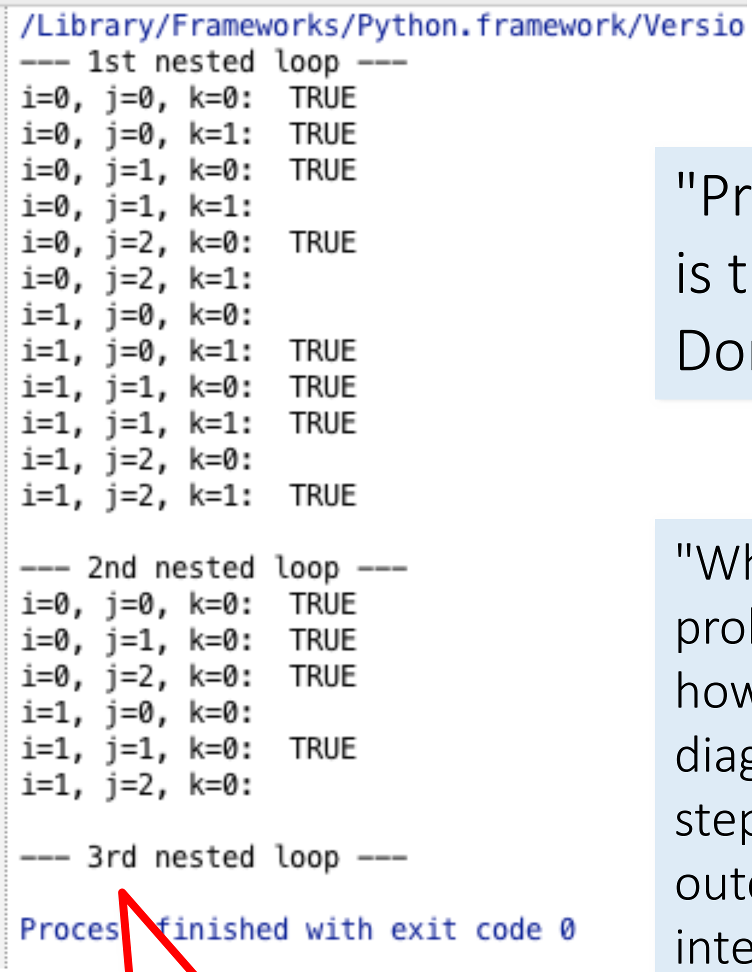
```
'''
Created on 3/23/2015
Illustrates nested Loop Join in SQL
__author__ = 'gatt'
'''

print "--- 1st nested loop ---"
for i in xrange(2):
    for j in xrange(3):
        for k in xrange(2):
            print "i=%d, j=%d, k=%d: " % (i, j, k),
            if i == j or i == k:
                print "TRUE",
            print

print "\n--- 2nd nested loop ---"
for i in xrange(2):
    for j in xrange(3):
        for k in xrange(1):
            print "i=%d, j=%d, k=%d: " % (i, j, k),
            if i == j or i == k:
                print "TRUE",
            print

print "\n--- 3rd nested loop ---"
for i in xrange(2):
    for j in xrange(3):
        for k in xrange(0):
            print "i=%d, j=%d, k=%d: " % (i, j, k),
            if i == j or i == k:
                print "TRUE",
            print

```



```
/Library/Frameworks/Python.framework/Versio
--- 1st nested loop ---
i=0, j=0, k=0:  TRUE
i=0, j=0, k=1:  TRUE
i=0, j=1, k=0:  TRUE
i=0, j=1, k=1:
i=0, j=2, k=0:  TRUE
i=0, j=2, k=1:
i=1, j=0, k=0:
i=1, j=0, k=1:  TRUE
i=1, j=1, k=0:  TRUE
i=1, j=1, k=1:  TRUE
i=1, j=2, k=0:
i=1, j=2, k=1:  TRUE

--- 2nd nested loop ---
i=0, j=0, k=0:  TRUE
i=0, j=1, k=0:  TRUE
i=0, j=2, k=0:  TRUE
i=1, j=0, k=0:
i=1, j=1, k=0:  TRUE
i=1, j=2, k=0:

--- 3rd nested loop ---

Proces  finished with exit code 0
```

The comparison gets never evaluated

> "Premature optimization is the root of all evil."
> Donald Knuth (1974)

> "When you are diagnosing problems, don't think about how you will solve them—just diagnose them. Blurring the steps leads to suboptimal outcomes because it interferes with uncovering the true problems."
> Ray Dalio (Principles, 2017)

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Quiz

Our colorful hands represent "team exercises"
If we are online, please make a screenshot!

302

Product (<u>pName</u>, price, category, manufacturer)
Company (<u>cName</u>, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

```
SELECT  DISTINCT cName
FROM
WHERE
```

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Quiz: Answer 1

Our colorful hands represent "team exercises"
If we are online, please make a screenshot!

Product (<u>pName</u>, price, category, manufacturer)
Company (<u>cName</u>, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

```
SELECT  DISTINCT cName
FROM    Product as P, Company
WHERE   country = 'USA'
    and P.price < 20
    and P.price > 25
    and P.manufacturer = cName
```

What about this query?

?

Quiz: Answer 1

Product (<u>pName</u>, price, category, manufacturer)
Company (<u>cName</u>, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

```
SELECT  DISTINCT cName
FROM    Product as P, Company
WHERE   country = 'USA'
   and  P.price < 20
   and  P.price > 25
   and  P.manufacturer = cName
```

Wrong! Gives empty result: There is no product with price <20 and >25

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Quiz: Answer 2

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

```
SELECT  DISTINCT cName
FROM    Product as P, Company
WHERE   country = 'USA'
    and (P.price < 20
    or  P.price > 25)
    and P.manufacturer = cName
```

What about this query?

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

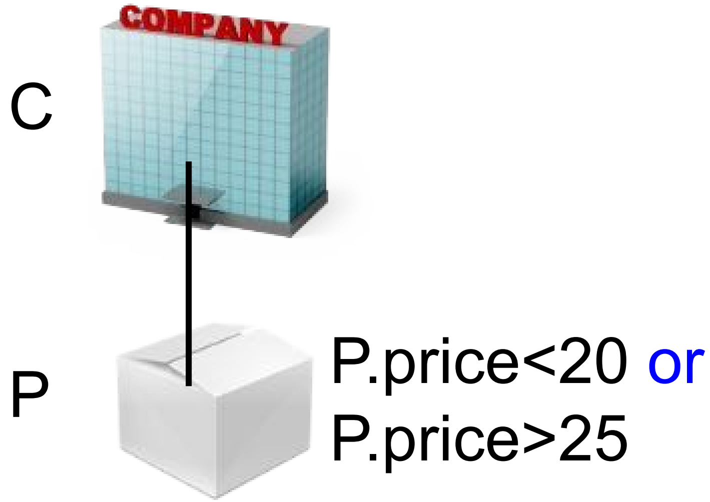
Quiz: Answer 2

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

Returns companies with single product w/price (<20 or >25)

```
SELECT  DISTINCT cName
FROM    Product as P, Company
WHERE   country = 'USA'
    and (P.price < 20
    or  P.price > 25)
    and P.manufacturer = cName
```

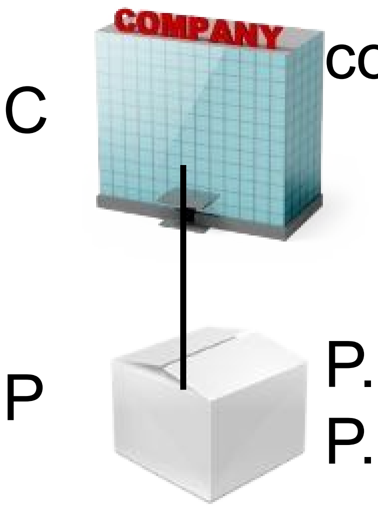


SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Quiz: Answer 1

Product (<u>pName</u>, price, category, manufacturer)
Company (<u>cName</u>, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.



country='USA'

P.price < 20 and / or
P.price > 25

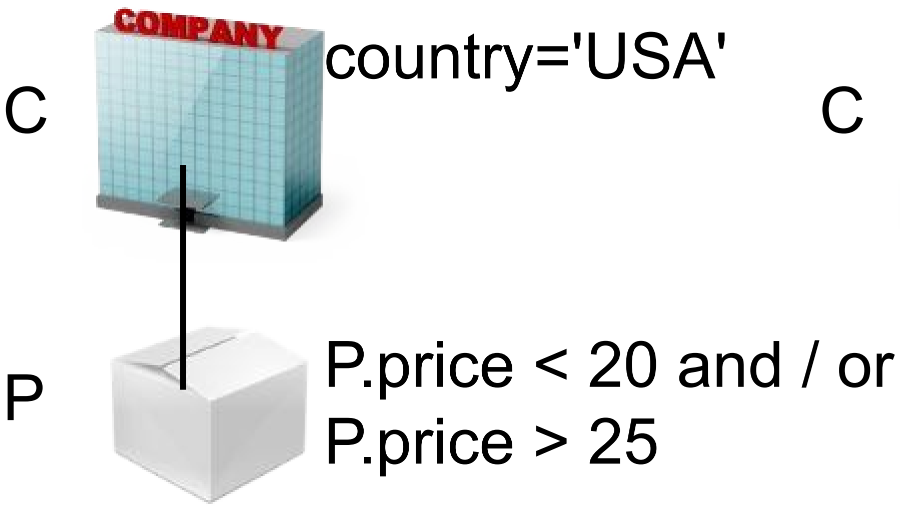
What do we actually want?

?

Quiz: Answer 1 vs. what we actually want

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

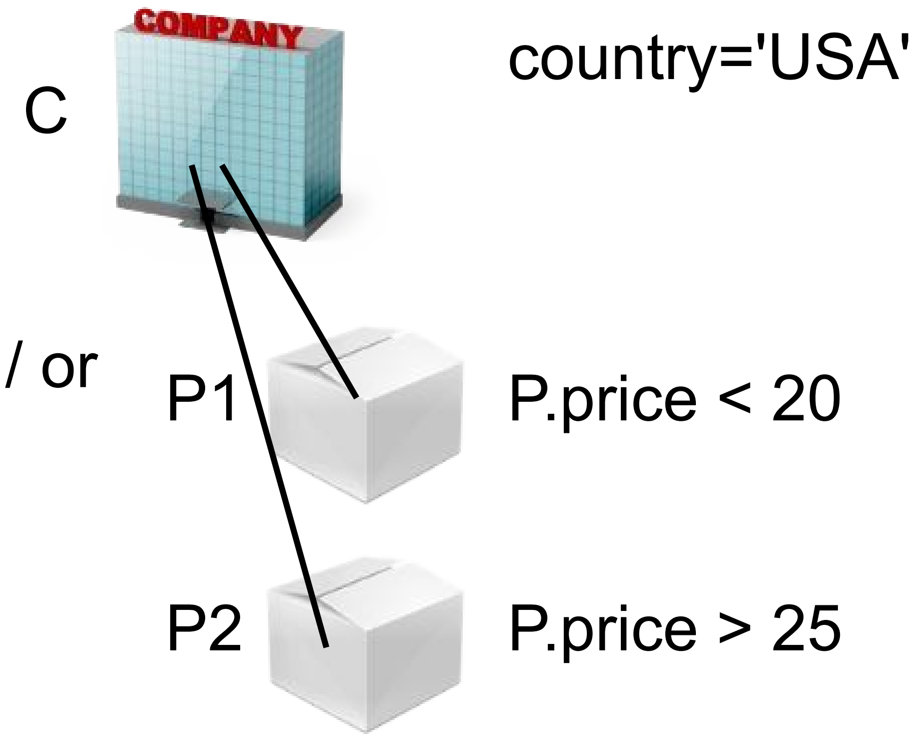


C

country='USA'

P

P.price < 20 and / or
P.price > 25



C

country='USA'

P1

P.price < 20

P2

P.price > 25

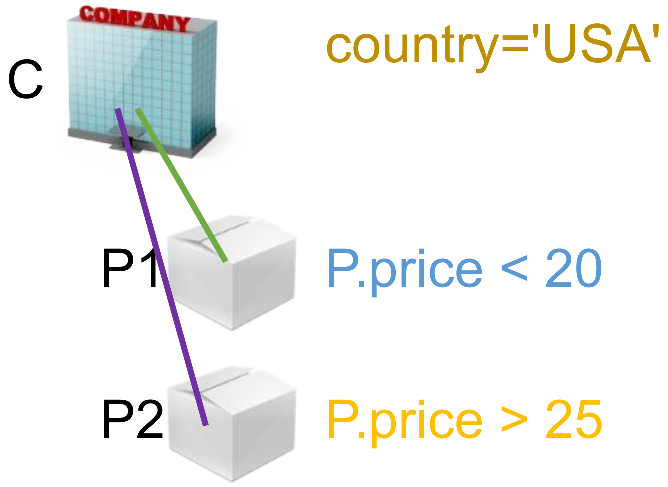
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Quiz: correct answer: we need "self-joins"!

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

```
SELECT  DISTINCT cName
FROM    Product as P1, Product as P2, Company
WHERE   country = 'USA'
    and P1.price < 20
    and P2.price > 25
    and P1.manufacturer = cName
    and P2.manufacturer = cName
```



Quiz response: we need "self-joins"!

P1

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

P2

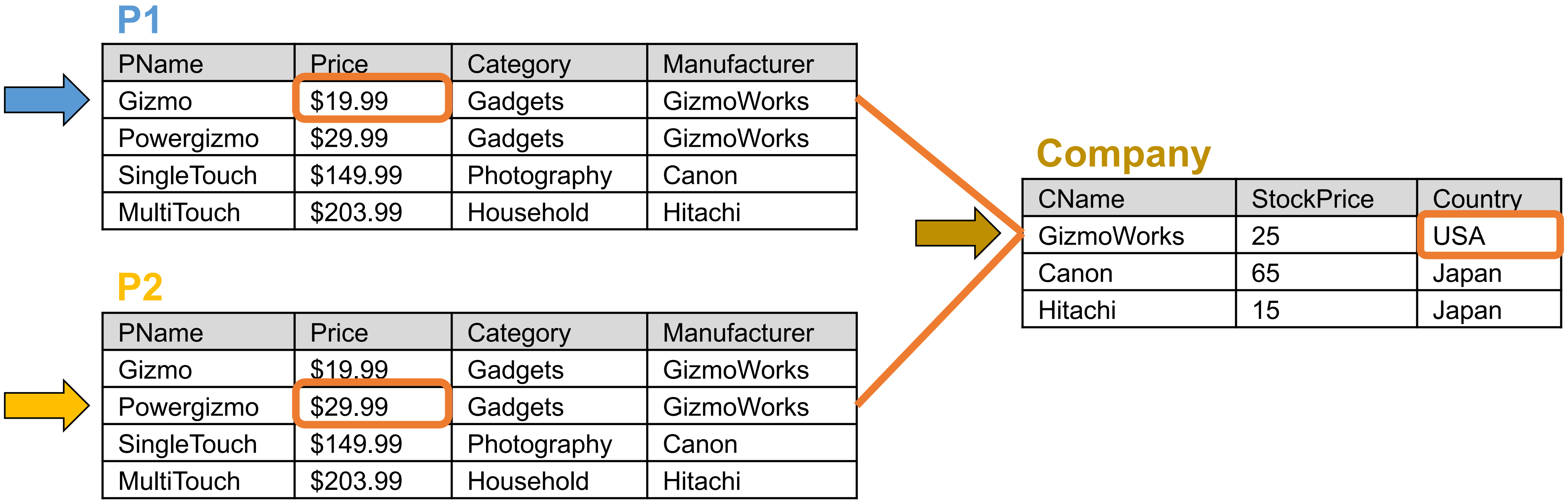
PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT  DISTINCT cName
FROM    Product as P1, Product as P2, Company
WHERE   country = 'USA'
    and P1.price < 20
    and P2.price > 25
    and P1.manufacturer = cName
    and P2.manufacturer = cName
```

Quiz response: we need "self-joins"!



P1

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

P2

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT  DISTINCT cName
FROM    Product as P1, Product as P2, Company
WHERE   country = 'USA'
    and P1.price < 20
    and P2.price > 25
    and P1.manufacturer = cName
    and P2.manufacturer = cName
```

CName
GizmoWorks

Outline: T1-U1: SQL

- SQL
  - Schema, keys, referential integrity
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Theta Joins
  - Nulls & Outer joins
  - Top-k
  - [Recursion: moved to T1-U4: Datalog]

Grouping and Aggregation

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

→ ?

Q: For each product, find Total Quantities (TQ = sum of quantities) purchased, for all products with price >1.

Grouping and Aggregation

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

⇨

Product	TQ
Bagel	?
Banana	?

Q: For each product, find Total Quantities (TQ = sum of quantities) purchased, for all products with price >1.

Grouping and Aggregation

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
~~Banana~~	~~1~~	~~50~~
Banana	2	10
Banana	4	10

Product	TQ
Bagel	40
Banana	20

Q: For each product, find Total Quantities (TQ = sum of quantities) purchased, for all products with price >1.

From → Where → Group By → Select

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
~~Banana~~	~~1~~	~~50~~
Banana	2	10
Banana	4	10

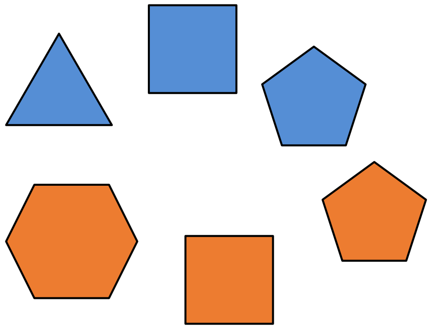
Product	TQ	PRICE
Bagel	40	
Banana	20	2,4
B	20	4

Select contains
- grouped attributes
- and aggregates

```
4  SELECT     product, sum(quantity) as TQ (     PRICE
1  FROM       Purchase
2  WHERE      price > 1
3  GROUP BY   product
```

Groupings illustrated with colored shapes

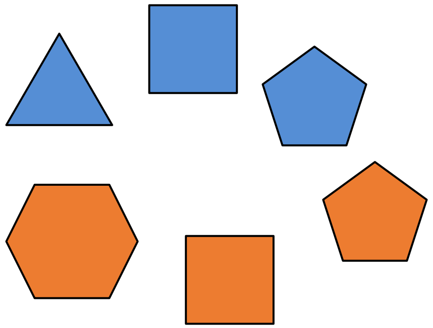
group by color



```
SELECT  color,
        avg(numc) anc
FROM    Shapes
GROUP BY color
```

⇨ ?

group by numc (# of corners)



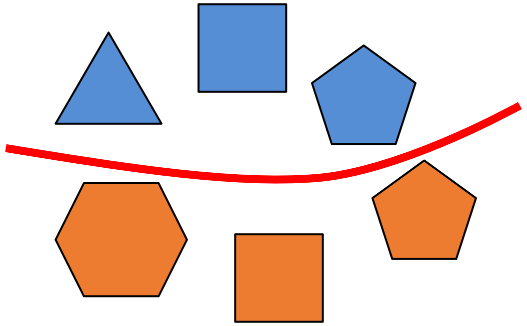
```
SELECT  numc
FROM    Shapes
GROUP BY numc
```

⇨ ?

Groupings illustrated with colored shapes

color	numc
blue	3
blue	4
blue	5
orange	4
orange	5
orange	6

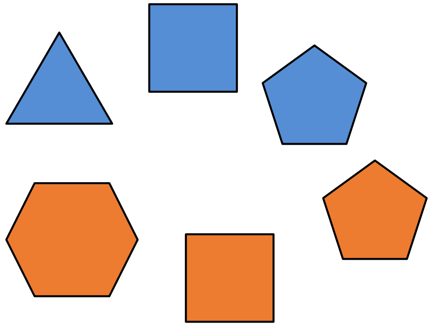
group by color



```
SELECT  color,
        avg(numc) anc
FROM    Shapes
GROUP BY color
```

⇨ ?

group by numc (# of corners)



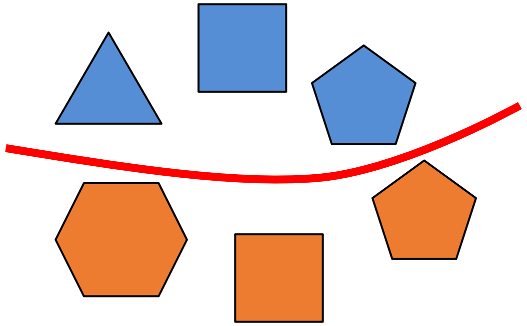
```
SELECT  numc
FROM    Shapes
GROUP BY numc
```

⇨ ?

Groupings illustrated with colored shapes

color	numc
blue	3
blue	4
blue	5
orange	4
orange	5
orange	6

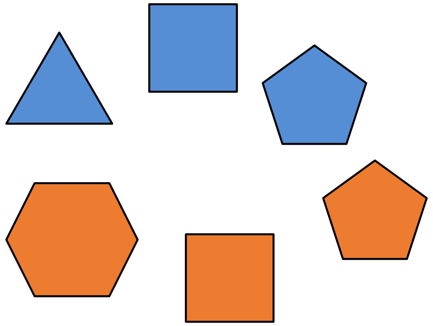
group by color



```
SELECT   color,
         avg(numc) anc
FROM     Shapes
GROUP BY color
```

color	anc
blue	4
orange	5

group by numc (# of corners)



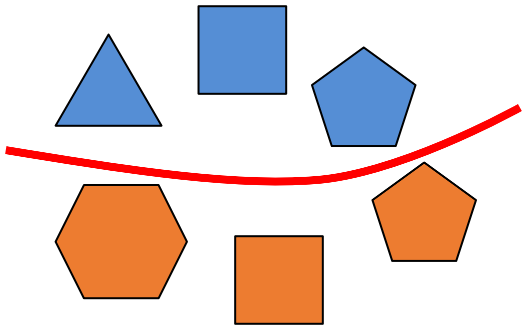
```
SELECT   numc
FROM     Shapes
GROUP BY numc
```

?

Groupings illustrated with colored shapes

color	numc
blue	3
blue	4
blue	5
orange	4
orange	5
orange	6

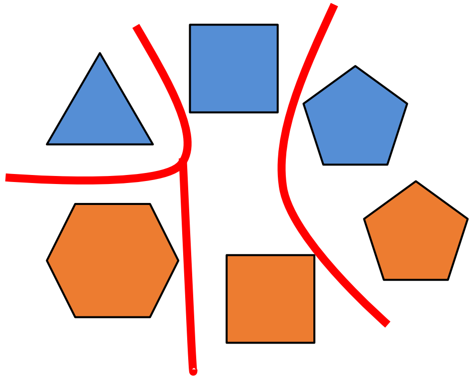
group by color



```
SELECT  color,
        avg(numc) anc
FROM    Shapes
GROUP BY color
```

color	anc
blue	4
orange	5

group by numc (# of corners)



```
SELECT  numc
FROM    Shapes
GROUP BY numc
```

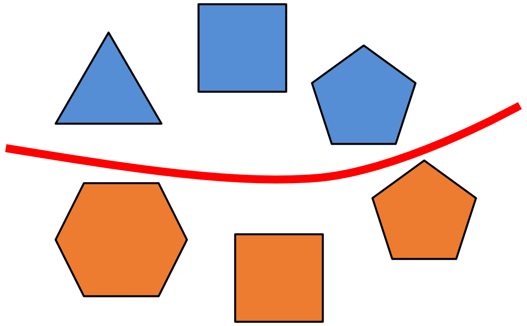
numc
3
4
5
6

Without group by ?

Groupings illustrated with colored shapes

color	numc
blue	3
blue	4
blue	5
orange	4
orange	5
orange	6

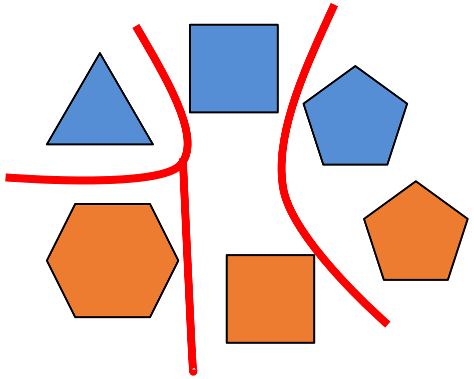
group by color



```
SELECT  color,
        avg(numc) anc
FROM    Shapes
GROUP BY color
```

color	anc
blue	4
orange	5

group by numc (# of corners)



```
SELECT  numc
FROM    Shapes
GROUP BY numc
```

numc
3
4
5
6

Same as:

```
SELECT  DISTINCT numc
FROM    Shapes
```

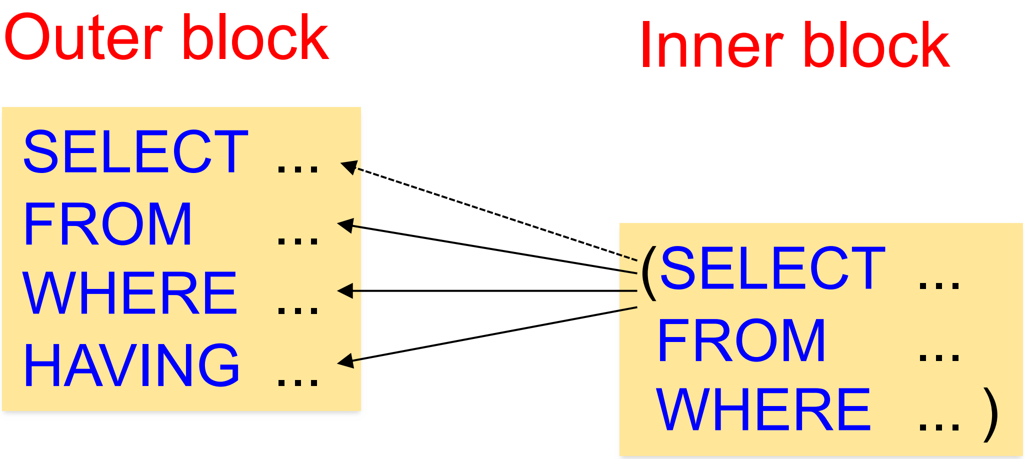
Without group by!

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Outline: T1-U1: SQL

- SQL
  - Schema, keys, referential integrity
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Theta Joins
  - Nulls & Outer joins
  - Top-k
  - [Recursion: moved to T1-U4: Datalog]

Subqueries = Nested queries



We focus mainly on nestings in the WHERE clause, which are the most expressive type of nesting.

- We can nest queries because SQL is compositional:
  - Input & Output are represented as relations (multisets)
  - Subqueries also return relations; thus the output of one query can thus be used as the input to another (nesting)
- This is extremely powerful (think in terms of input/output)
- A complication: subqueries can be correlated (not just in-/output)

Subqueries in

SELECT clause

FROM clause (also called "derived tables")

WHERE clause

HAVING clause

Subqueries in FROM clause = Derived tables

308

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

Product	TQ
Bagel	40
Banana	70

MTQ
70

Q1: For each product, find total quantities (sum of quantities) purchased.

Q2: Find the maximal total quantities purchased across all products.

```
SELECT product, SUM(quantity) as TQ
FROM Purchase
GROUP BY   product
```

?

Subqueries in FROM clause = Derived tables


308

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

→

X

Product	TQ
Bagel	40
Banana	70

→

MTQ
70

Q1: For each product, find total quantities (sum of quantities) purchased.

Q2: Find the maximal total quantities purchased across all products.

(

```
SELECT product, SUM(quantity) as TQ
FROM Purchase
GROUP BY   product
```

) X

?

Subqueries in FROM clause = Derived tables

308

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

→

X

Product	TQ
Bagel	40
Banana	70

→

MTQ
70

Q1: For each product, find total quantities (sum of quantities) purchased.

```
( SELECT product, SUM(quantity) as TQ
  FROM Purchase
  GROUP BY   product )  X
```

Q2: Find the maximal total quantities purchased across all products.

```
SELECT MAX(TQ) as MTQ
FROM X
```

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Subqueries in FROM clause = Derived tables

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

```
SELECT MAX(TQ) as MTQ
FROM (SELECT product, SUM(quantity) as TQ
      FROM Purchase
      GROUP BY product) X
```

MTQ
70

Q1: For each product, find total quantities (sum of quantities) purchased.

```
SELECT product, SUM(quantity) as TQ
FROM Purchase
GROUP BY   product
```

Q2: Find the maximal total quantities purchased across all products.

```
SELECT MAX(TQ) as MTQ
FROM X
```

Common Table Expressions (CTE): WITH clause

308

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

```
SELECT MAX(TQ) as MTQ
FROM (SELECT product, SUM(quantity) as TQ
        FROM Purchase
        GROUP BY product) X
```

MTQ
70

CTE (Common Table Expression)

```
WITH  X as
        (SELECT product, SUM(quantity) as TQ
        FROM Purchase
        GROUP BY product)
```

Query using CTE

```
SELECT MAX(TQ) as MTQ
FROM X
```

The WITH clause defines a temporary relation that is available only to the query in which it occurs. Sometimes easier to read. Very useful for queries that need to access the same intermediate result multiple times

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Subqueries in

SELECT clause

FROM clause

WHERE clause (including IN, ANY, ALL)

HAVING clause

Subqueries in WHERE clause

What do these queries return?

R

a
1
2

W

a	b
2	0
3	0
4	0

```
SELECT  a
FROM    R
WHERE   a IN
        (SELECT a FROM W)
```

⇨ ?

```
SELECT  a
FROM    R
WHERE   a < ANY
        (SELECT a FROM W)
```

⇨ ?

```
SELECT  a
FROM    R
WHERE   a < ALL
        (SELECT a FROM W)
```

⇨ ?

Subqueries in WHERE clause

305

R

a
1
2

W

a	b
2	0
3	0
4	0

What do these queries return?

```
SELECT   a
FROM     R
WHERE    a IN
         (SELECT a FROM W)
```

a
2

Since 2 is in the set (bag) (2, 3, 4)

```
SELECT   a
FROM     R
WHERE    a < ANY
         (SELECT a FROM W)
```

?

```
SELECT   a
FROM     R
WHERE    a < ALL
         (SELECT a FROM W)
```

?

Subqueries in WHERE clause

R

a
1
2

W

a	b
2	0
3	0
4	0

What do these queries return?

```
SELECT   a
FROM     R
WHERE    a IN
         (SELECT a FROM W)
```

a
2

Since 2 is in the set (bag) (2, 3, 4)

```
SELECT   a
FROM     R
WHERE    a < ANY
         (SELECT a FROM W)
```

a
1
2

Since 1 and 2 are < than at least one ("any") of 2, 3 or 4

```
SELECT   a
FROM     R
WHERE    a < ALL
         (SELECT a FROM W)
```

?

Subqueries in WHERE clause

305

R

a
1
2

W

a	b
2	0
3	0
4	0

What do these queries return?

SQLlite does not support "ANY" or "ALL" ☹

```
SELECT   a
FROM     R
WHERE    a IN
         (SELECT a FROM W)
```

a
2

Since 2 is in the set (bag) (2, 3, 4)

```
SELECT   a
FROM     R
WHERE    a < ANY
         (SELECT a FROM W)
```

a
1
2

Since 1 and 2 are < than at least one ("any") of 2, 3 or 4

```
SELECT   a
FROM     R
WHERE    a < ALL
         (SELECT a FROM W)
```

a
1

Since 1 is < than each ("all") of 2, 3, and 4

Correlated subqueries

- In all previous cases, the nested subquery in the inner select block could be entirely evaluated before processing the outer select block.
  - Recall the "compositional" nature of relational queries
  - This is no longer the case for correlated nested queries.
- Whenever a condition in the WHERE clause of a nested query references some column of a table declared in the outer query, the two queries are said to be correlated.
  - The nested query is then evaluated once for each tuple (or combination of tuples) in the outer query (that's the conceptual evaluation strategy)

Correlated subquery (existential ∃)

316

slightly different product database!

Product

PName	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

cid	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	Canon	65	Japan
3	Hitachi	15	Japan

Q_1: Find all companies that make some product(s) with price < 25

Using IN: Set / Bag membership

```
SELECT  DISTINCT C.cname
FROM    Company C
WHERE   C.cid IN ( SELECT  P.cid
                   FROM    Product P
                   WHERE   P.price < 25)
```

Is this a correlated nested query ?

Correlated subquery (existential ∃)

316

Product

PName	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

cid	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	Canon	65	Japan
3	Hitachi	15	Japan

slightly different product database!

Q_1: Find all companies that make some product(s) with price < 25

Using IN: Set / Bag membership

```
SELECT  DISTINCT C.cname
FROM    Company C
WHERE   C.cid IN ( SELECT  P.cid
                   FROM    Product P
                   WHERE   P.price < 25)
```

Not a correlated nested query!

```
SELECT  DISTINCT C.cname
FROM    Company C
WHERE   C.cid IN ( 1, 2 )
```

Inner query does not reference outer query! You could first evaluate the inner query by itself.

Correlated subquery (existential ∃)

Product

<u>PName</u>	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

<u>cid</u>	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	Canon	65	Japan
3	Hitachi	15	Japan

Q_1: Find all companies that make <u>some</u> product(s) with price < 25

Using EXISTS: TRUE if the subquery's result is NOT empty

```
SELECT  DISTINCT C.cname
FROM    Company C
WHERE   EXISTS ( SELECT  *
                 FROM    Product P
                 WHERE   P.cid = C.cid
                 and     P.price < 25)
```

Is this a correlated nested query ?

Correlated subquery (existential ∃)

Product

PName	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

cid	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	Canon	65	Japan
3	Hitachi	15	Japan

Q_1: Find all companies that make some product(s) with price < 25

Using EXISTS: TRUE if the subquery's result is NOT empty

```
SELECT  DISTINCT C.cname
FROM    Company C
WHERE   EXISTS ( SELECT  *
                 FROM    Product P
                 WHERE   P.cid = C.cid
                 and     P.price < 25)
```

This is a correlated nested query! Notice the additional join condition referencing a relation from the outer query.

Recall our conceptual evaluation strategy!

Correlated subquery (existential ∃)

Product

<u>PName</u>	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

<u>cid</u>	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	Canon	65	Japan
3	Hitachi	15	Japan

Q_1: Find all companies that make <u>some</u> product(s) with price < 25

Using ANY (also SOME): again set / bag comparison

```
SELECT  DISTINCT C.cname
FROM    Company C
WHERE   25 > ANY ( SELECT  price
                   FROM    Product P
                   WHERE   P.cid = C.cid)
```

But do we really need to write this query as nested query

?

SQLlite does not support "ANY" ☹

Correlated subquery (existential ∃)

316

Product

PName	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

cid	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	Canon	65	Japan
3	Hitachi	15	Japan

Q_1: Find all companies that make some product(s) with price < 25

```
SELECT  DISTINCT C.cname
FROM    Company C, Product P
WHERE   C.cid = P.cid
and     P.price < 25
```

We did not need to write nested queries;
we can "unnest" it!

Existential quantifiers are easy ☺

Correlated subquery (universal ∀)

316

Product

PName	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

cid	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	Canon	65	Japan
3	Hitachi	15	Japan

30

~~Q_1: Find all companies that make some product(s) with price < 25~~

Q_2: Find all companies that make only products with price < 25

≡ Q_2: Find all companies for which all products have price < 25

≡ Q_2: Find all companies that do not have any product with price >= 25

Universal quantifiers are more complicated ! ☹
(Think about the companies that should not be returned)

All three formulations are equivalent: a company with no product will be returned!

Correlated subquery (universal ∀ = not exists ∄)

316

Q_2: Find all companies that make only products with price < 25

Step 1: Q_2': Find the other companies that make some product(s) with price ≥ 25

```
SELECT DISTINCT C.cname
FROM   Company C
WHERE  C.cid IN       ( SELECT P.cid
                        FROM   Product P
                        WHERE  P.price >= 25)
```

First think about the companies that should not be returned!

Step 2: Q_2: Find all companies that make no products with price ≥ 25

```
SELECT DISTINCT C.cname
FROM   Company C
WHERE  C.cid NOT IN ( SELECT P.cid
                      FROM   Product P
                      WHERE  P.price >= 25)
```

Correlated subquery (universal ∀ = not exists ∄)

Q_2: Find all companies that make only products with price < 25

Step 1: Q_2': Find the other companies that make some product(s) with price ≥ 25

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE   EXISTS       ( SELECT  *
                       FROM    Product P
                       WHERE   C.cid = P.cid
                       and     P.price >= 25)
```

First think about the companies that should not be returned!

Step 2: Q_2: Find all companies that make no products with price ≥ 25

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE   NOT EXISTS ( SELECT  *
                     FROM    Product P
                     WHERE   C.cid = P.cid
                     and     P.price >= 25)
```

Correlated subquery (universal ∀ = not exists ∄)

316

Q_2: Find all companies that make only products with price < 25

Step 1: Q_2': Find the other companies that make some product(s) with price ≥ 25

```
SELECT DISTINCT C.cname
FROM     Company C
WHERE  25 <= ANY     ( SELECT  P.price
                       FROM      Product P
                       WHERE   C.cid = P.cid)
```

First think about the companies that should not be returned!

Step 2: Q_2: Find all companies that make no products with price ≥ 25

```
SELECT DISTINCT C.cname
FROM     Company C
WHERE  25 > ALL        ( SELECT  P.price
                         FROM      Product P
                         WHERE   C.cid = P.cid)
```

A natural question

Q_2: Find all companies that make only products with price < 25

- How can we unnest (no GROUP BY) the universal quantifier query ?

```
SELECT  ...
FROM    ...
WHERE   ...
```

?

Source: Dan Suciu