From Complexity to Clarity

A Journey Towards Streamlined and Scalable Integrations

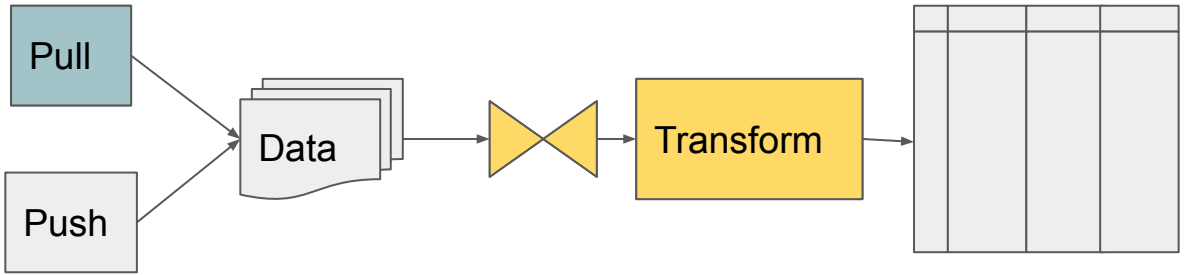
Context - What is AppOmni?

- Help customers secure their SaaS apps
- Visibility
- Expert recommendations
- Configuration Policies
- Much more

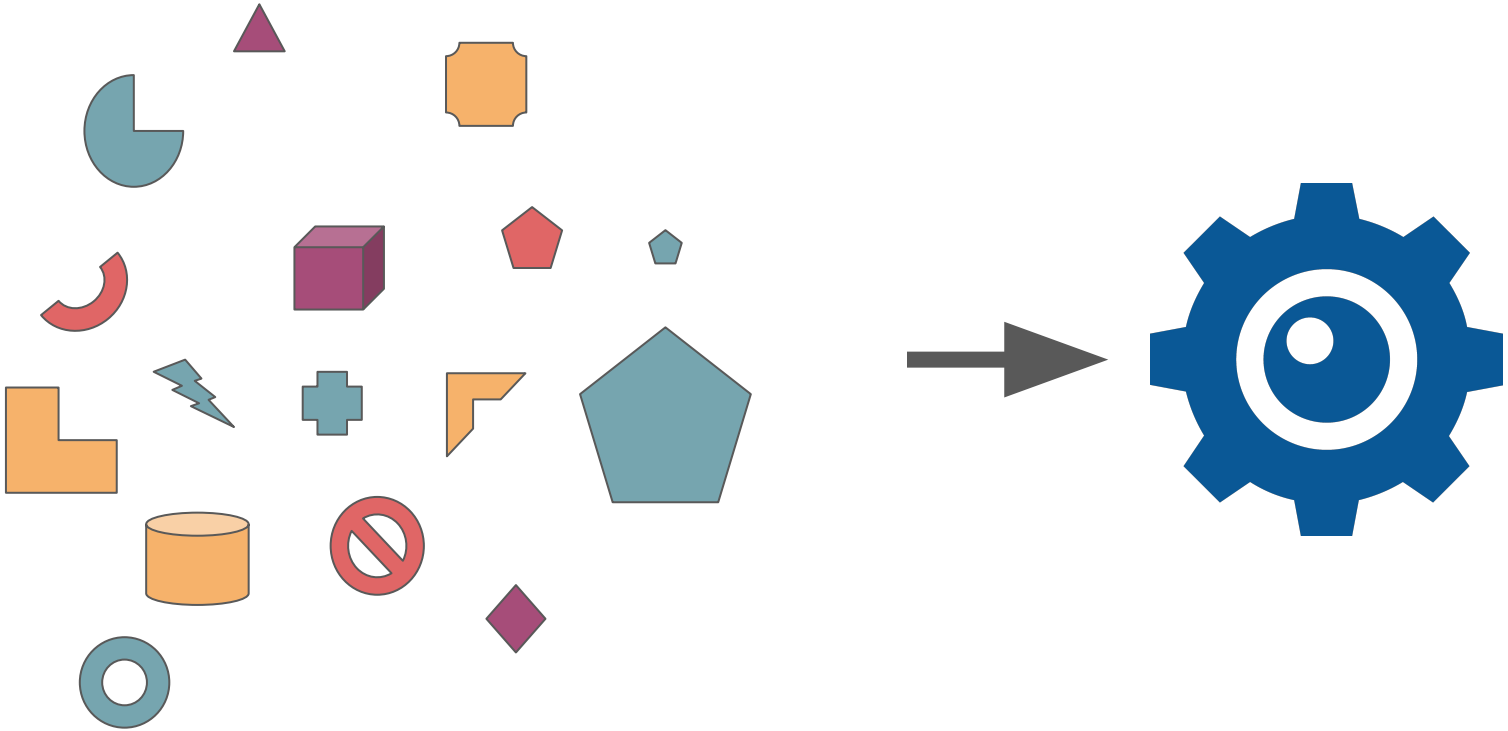
How do we do this?

- Customers provide AppOmni access to their SaaS tenants.
- We collect relevant configuration data.
- Create view of the current state of the service.
- We provide features on top of that view.

Data Ingest Pipeline



Target Services



Example

```
user_list -> [..., {“id”: 1, “name”: “sam”}, ...]

user_details -> [..., {“id”: 1, “email”: “sam@shire.org”}, ...]

user_settings -> [..., {“email”: “sam@shire.org”, “mfa_is_setup”: True}, ...]


[..., User(api_id=1, email=”sam@shire.org”, name=”sam”, has_mfa=True), ...]
```

This sucks, but why?

user_list

```
api_id_to_data: Dict[str, Dict] = {}  # hash table
retry_count = 0
while True:
    if 3 < retry_count:
        raise ValueError("No more retires!")
    try:
        resp = requests.get(...)
    except Exception:
        retry_count += 1
        time.sleep(retry_count ** 2)
    else:
        if not resp.ok:
            retry_count += 1
            time.sleep(retry_count ** 2)
        else:
            retry_count = 0
            page = resp.data()
            for user_data in page['users']:
                api_id = user_data['id']
                api_id_to_data[api_id] = {
                    'api_id': user_data['id'],
                    'name': user_data['user_name']
                }
```

Request Page

Proc response

Stash in dict for ref

user_detail

```
email_to_details: Dict[str, Dict] = {}
retry_count = 0
while True:
    if 3 < retry_count:
        raise ValueError("No more retires!")
    try:
        resp = requests.get(..., params={"user_id": user_id})
    except Exception:
        retry_count += 1
        time.sleep(retry_count ** 2)
    else:
        if not resp.ok:
            retry_count += 1
            time.sleep(retry_count ** 2)
        else:
            retry_count = 0

            details = resp.data()
            api_id = details['id']
            user_dict = api_id_to_data[api_id]
            email = user_dict['user_email']
            email_to_details[email] = dict(
                **user_dict,
                email=email,
            )
```

Request Page

Proc response

Access prior

Stash in dict for ref

user_settings

```
users: Set[User] = set()
retry_count = 0
while True:
    if 3 < retry_count:
        raise ValueError("No more retires!")
    try:
        resp = requests.get(..., params={"user_id": user_id})
    except Exception:
        retry_count += 1
        time.sleep(retry_count ** 2)
    else:
        if not resp.ok:
            retry_count += 1
            time.sleep(retry_count ** 2)
        else:
            retry_count = 0

            settings = resp.data()
            email = settings['user_email']
            user_dict = email_to_details[email]
            users.add(User(
                **user_dict,
                has_mfa=settings['mfa_is_setup']
            ))
```

Request Page

Proc response

Access prior

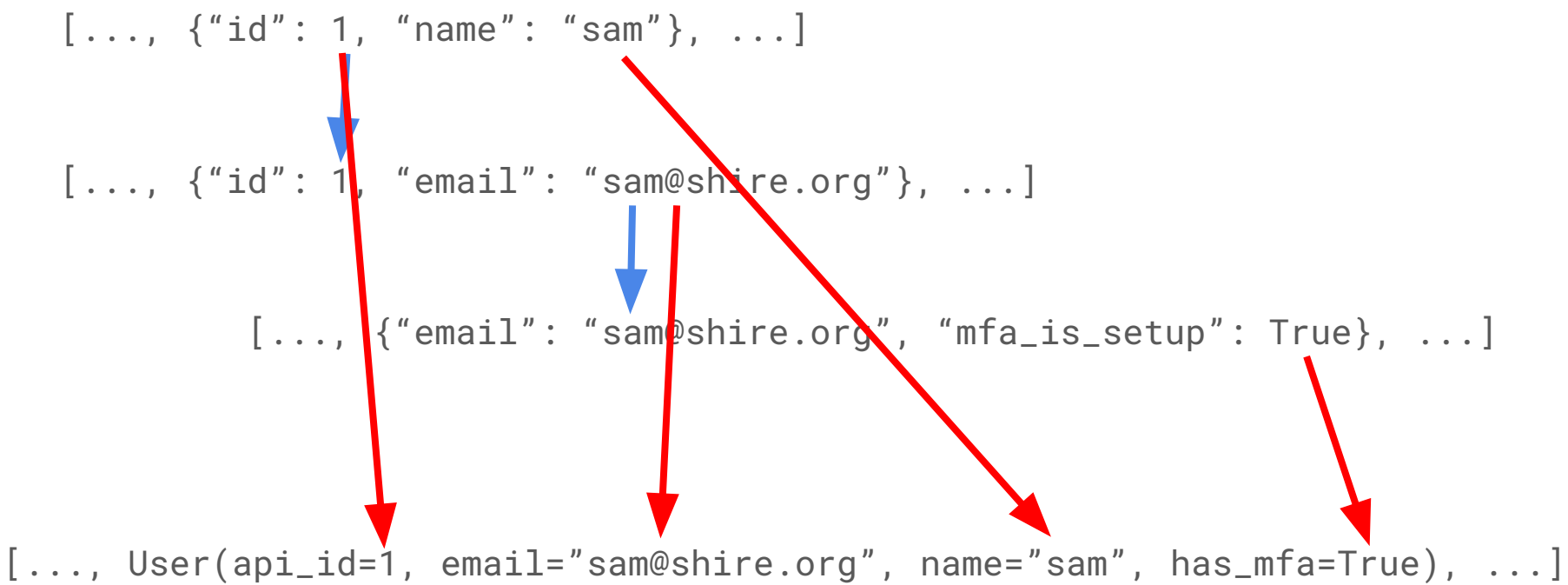
Better?

```
requester = Requester(...)

user_list_pages = requester.list(...)
api_id_to_data: Dict[str, Dict] = {}
for page in user_list_pages:
    for user_data in page['users']:
        api_id = user_data['id']
        api_id_to_data[api_id] = {
            'api_id': user_data['id'],
            'name': user_data['user_name']
        }


email_to_details: Dict[str, Dict] = {}
for usr_id in api_id_to_data:
    details = requester.get(..., params={"user_id": usr_id})
    api_id = details['id']
    user_dict = api_id_to_data[api_id]
    email = user_dict['user_email']
    email_to_details[email] = dict(
        **user_dict,
        email=email,
    )


users: Set[User] = set()
for usr_email in email_to_details:
    settings = requester.get(..., params={"user": usr_email})
    email = settings['user_email']
    user_dict = email_to_details[email]
    users.add(User(
        **user_dict,
        has_mfa=settings['mfa_is_setup']
    ))
```

Problems

- Evolvability of the platform.
- Throughput of new integrations.
- The definition of an integration tells us more about our ORM/http client/etc. than it does does about the target service.

What are we actually doing?



"I See What You Mean" by Peter Alvaro

- Queries, useful for more than just databases
- Data independence
  - Separation of logical representation from physical implementation

miniKanren

```
>>> from kanren import var, eq, run
>>> api_id = var()
>>> run(0, api_id, eq(api_id, 1))
(1, )
```

```
>>> run(0, api_id, eq(api_id, 1), eq(api_id, 5))
()
```

```
>>> name = var()
>>> run(0, [api_id, name], eq(api_id, 1), eq(name, "sam"))
([1, "sam"],)

# what is going on?
>>> run(
...     0,                              # number of solutions to find. 0 = all
...     [api_id, name],                 # The structure of the solutions.
...     eq(api_id, 1), eq(name, "sam"),  # “goals” that must be met.
... )
([1, "sam"],)                           # The solutions.
```

```
>>> run(0, [api_id, name], eq(api_id, 1))
([1, ~_2],)
```

~_2, since `name` remains “fresh”

```
>>> run(0, [api_id, name], eq([api_id, name], [1, "sam"]))
([1, "sam"],)
```

```
>>> run(0,
...     [api_id, name],
...     eq({"id": api_id, "user_name": name},
...        {"id": 1, "user_name": "sam"}),
... )
([1, "sam"],)
```

```
>>> run(0, {"api_id": api_id, "name": name},
...     eq({"id": api_id, "user_name": name},
...        {"id": 1, "user_name": "sam"}),
... )
({'api_id': 1, 'name': 'sam'},)
```

```
>>> email = var()
>>> run(0, {"api_id": api_id, "name": name, "email": email},
...     eq({"id": api_id, "user_name": name},
...        {"id": 1, "user_name": "sam"}),
...     eq({"id": api_id, "email": email},
...        {"id": 1, "email": "sam@shire.org"}),
... )
({'api_id': 1, 'name': 'sam', 'email': 'sam@shire.org'},)
```

```
>>> has_mfa = var()
>>> run(0, {"api_id": api_id, "name": name, "email": email, "has_mfa": has_mfa},
...     eq({"id": api_id, "user_name": name},
...        {"id": 1, "user_name": "sam"}),
...     eq({"id": api_id, "email": email},
...        {"id": 1, "email": "sam@shire.org"}),
...     eq({"id": api_id, "mfa_is_setup": has_mfa},
...        {"id": 1, "mfa_is_setup": True}),
... )
({'api_id': 1, 'name': 'sam', 'email': 'sam@shire.org', 'has_mfa': True},)
```

This space reserved for

PIZZA STAINS!

Review

```
>>> api_id, name, email, has_mfa = var(), var(), var(), var()
>>> user_list_data = {"id": 1, "user_name": "sam"}
>>> user_detail_data = {"id": 1, "email": "sam@shire.org"}
>>> user_settings_data = {"id": 1, "mfa_is_setup": True}
>>> run(0, {"api_id": api_id, "name": name, "email": email, "has_mfa": has_mfa},
...     eq({"id": api_id, "user_name": name},
...        user_list_data),
...     eq({"id": api_id, "email": email},
...        user_detail_data),
...     eq({"id": api_id, "mfa_is_setup": has_mfa},
...        user_settings_data),
... )
({'api_id': 1, 'name': 'sam', 'email': 'sam@shire.org', 'has_mfa': True},)
```

Logic Variables

Input data

Solution Shape

Goals

Solution

Beginnings of a Solution

```
>>> def process_user_list_resp(resp_data: Dict) -> List[User]:
...     ...

>>> def process_user_detail_resp(resp_data: Dict) -> List[User]:
...     ...

>>> def process_user_setting_resp(resp_data: Dict) -> List[User]:
...     ...

>>> rels = (
...     Relations(User)
...     .define(user_list, user_detail, on="api_id")
...     .define(user_detail, user_setting, on="email")
... )
```

Beginnings of a Solution

It's ~ inspired by miniKanren

```
User(api_id=..., email=..., name=..., has_mfa=...)
>>> user_list_data = [{"id": 1, "user_name": "sam"}, ]
>>> user_detail_data = [{"id": 1, "email": "sam@shire.org"}, ]
>>> user_setting_data = [{"email": "sam@shire.org", "mfa_is_setup": True}, ]
>>> rels = (
...     Relations(User)
...     .define(user_list, user_detail, on="api_id")
...     .define(user_detail, user_setting, on="email")
... )
>>> (LeftJoin(rels)
...  .add_data(user_list, proc_list_resp(user_list_data))
...  .add_data(user_detail, proc_detail_resp(user_detail_data))
...  .add_data(user_setting, proc_setting_resp(user_setting_data))
...  .results()
... )
[User(api_id=1, name="sam", email="sam@shire.org", has_mfa=True), ]
```

Logic Variables

Input data

Solution Shape

Goals

~ `run(...)`

Solution

Small black boxes only

```
>>> def process_user_list_resp(resp_data: Dict) -> List[User]:
...     contribs = set()
...     for user_dict in resp_data["users"]:
...         contribs.add(
...             User(
...                 api_id=user_dict["id"],
...                 name=user_dict["user_name"],
...             ))
...     return contribs

>>> user_list_page = [{"id": 1, "user_name": "sam"}, ]
>>> process_user_list_resp(user_list_page)
[User(api_id=1, name="sam"), ]
```

Something missing

- miniKanren was useful inspiration
- Still need “off the shelf” data model
- guidance around structures and operations

Relational Model

```
{
    User(api_id=1, name="sam"),
    ...,
}
```

User	
api_id	name
1	"sam"
...	...

```
>>> def process_user_list_resp(resp
...     ...

>>> def process_user_detail_resp(re
...     ...

>>> def process_user_setting_resp(r
...     ...

>>> rels = (
...     Relations(User)
...     .define(user_list, user_det
...     .define(user_detail, user_s
... )
>>> (LeftJoin(rels)
...  ...
...  .results()
... )
```

```
WITH user_list AS (
  SELECT id as api_id, user_name as name FROM user_list_resp
), user_detail AS (
  SELECT id as api_id, user_email as email FROM user_detail_resp
), user_setting AS (
  SELECT user_email as email, mfa_is_setup AS has_mfa FROM user_detail_resp
)
SELECT
  DITul.api_id, COALESCE(ud.email, 'nil'), ul.name COALESCE(uds.has_mfa, False
  FROM
    user_list ul
  LEFT JOIN
  (SELECT
      ud.api_id, ud.email, COALESCE(us.has_mfa, False)
      FROM user_detail ud LEFT JOIN user_setting us USING email
  ) uds USING api_id;
```

What properties can we provide, generally?

- Going back to “I see what you mean”
- The potential of query languages to express distributed systems problems.
- We have a distributed systems problem, what guarantees can we make?

Keeping CALM: When Distributed Consistency is Easy

By Joseph Hellerstein and Peter Alvaro

Keeping CALM

- Description of the CALM Theorem.
- “Does my program produce deterministic outcomes despite non-determinism in the runtime system?”
- Monotonic programs produce consistent outcomes (for the same input)
- When is coordination required?

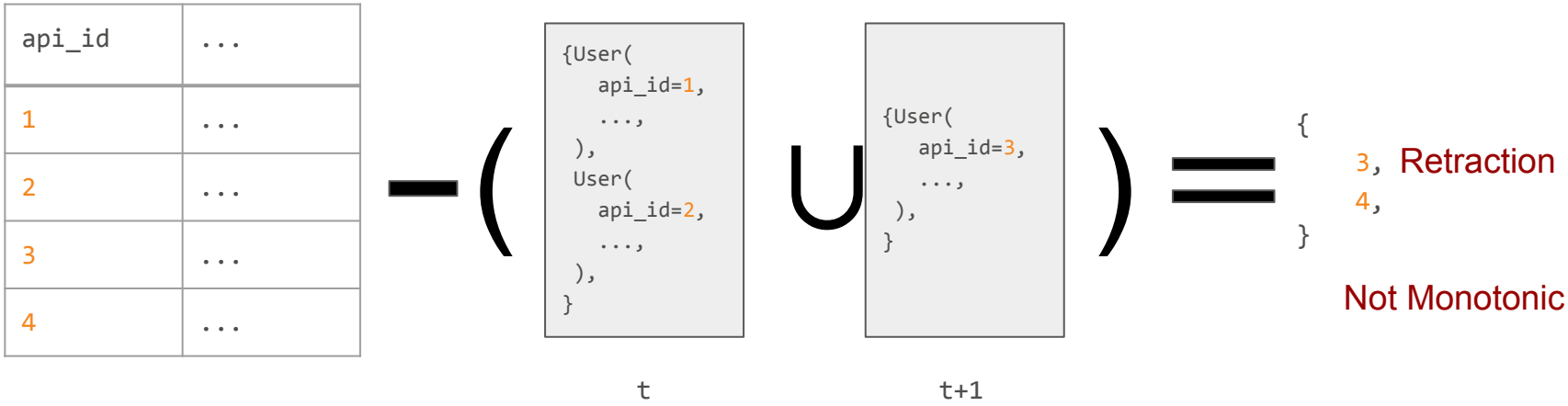
How does that help us

How does that help us

```
User(api_id=1, name="sam")

User(api_id=1, email="sam@shire.org")

User(email="sam@shire.org", has_mfa=True)
```

```
User(
    api_id=1,
    email="sam@shire.org",
    name="sam",
    has_mfa=True,
)
```

What does that mean for us? - View Maintenance



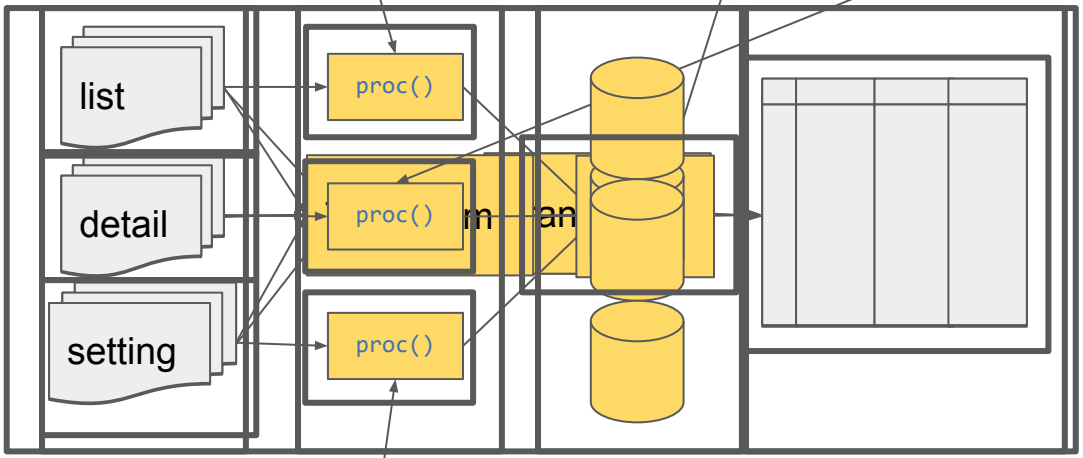
Is it better?

```
>>> def process_user_list_resp(resp_data: Dict) -> List[User]:
...     ...

>>> def process_user_detail_resp(resp_data: Dict) -> List[User]:
...     ...

>>> def process_user_setting_resp(resp_data: Dict) -> List[User]:
...     ...

>>> rels = (
...     Relations(User)
...     .define(user_list, user_detail, on="api_id")
...     .define(user_detail, user_setting, on="email")
... )
```

```
(Relations(User)
 .define(user_list, user_detail, on="api_id")
 .define(user_detail, user_setting, on="email"))
```

process_user_list_resp()

process_user_detail_resp()



process_user_setting_resp()

Is it better?

- Evolvability of the platform? ✓
- Throughput of new integrations? ✓
- ~~The definition of an integration tells us more about our ORM/http client/etc. than it does does about the target service.~~

  Data independence? ~✓

What do the Humans think?

- Picked it up quickly
- Reception is positive
- Over delivered

References

- Keeping CALM: When Distributed Consistency is Easy by Joseph M. Hellerstein, Peter Alvaro
- The Reasoned Schemer (The MIT Press) 2nd ed. Edition by Daniel P. Friedman (Author), William E. Byrd (Author), Oleg Kiselyov (Author)
- "I See What You Mean" by Peter Alvaro, Strange Loop Conference

Additional Resources

- SQL and Relational Theory, 3rd Edition by C.J. Date
- A CRDT Primer Part I and II by John Mumm
- Marc Shapiro, Nuno Preguiça, Carlos Baquero, Marek Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types.
- Introduction to Lattices and Order by B.A. Davey and H.A. Priestley

Questions?

We are hiring!

GitHub: steven-cutting <- Slides will be here

LinkedIn: stevencutting

scutting@appomni.com