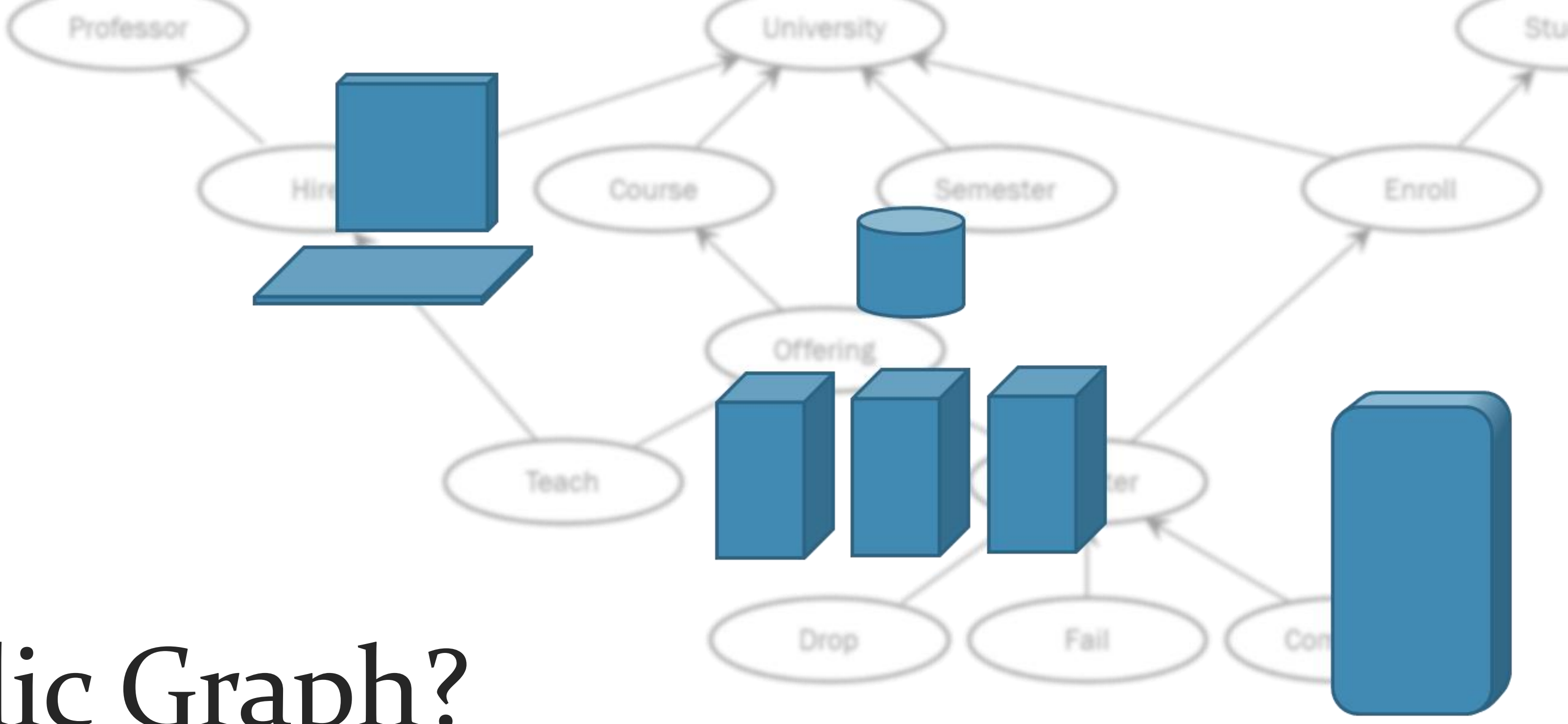
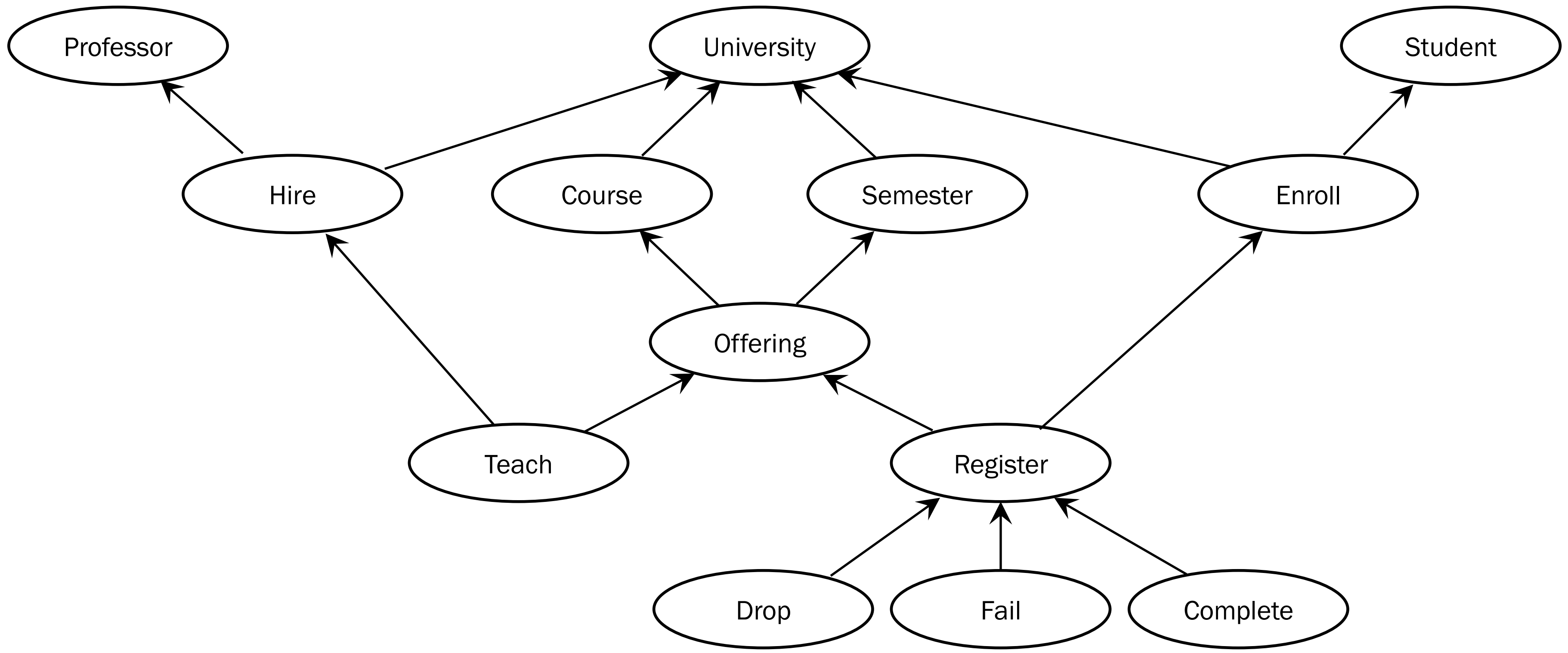
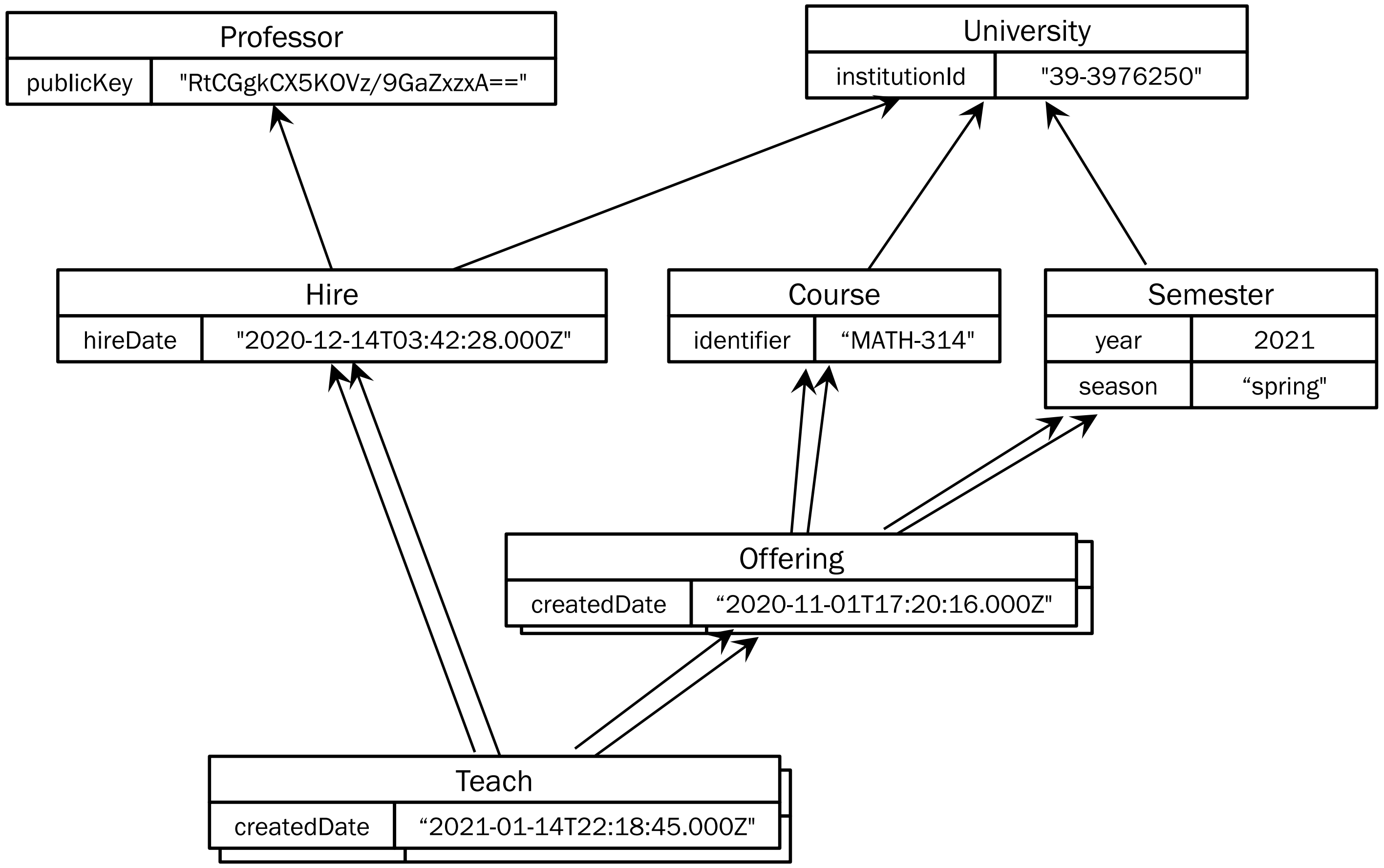


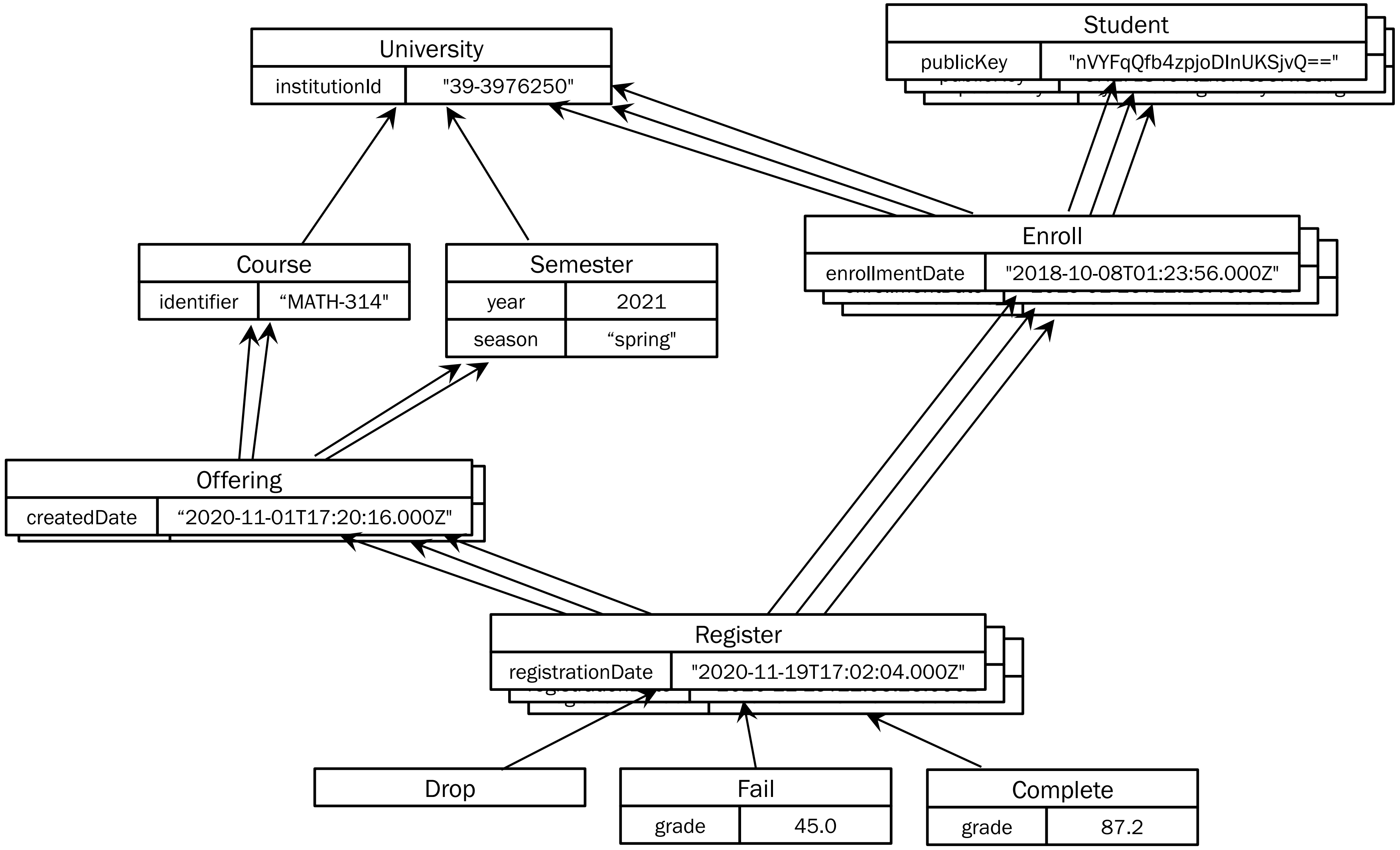
Directed Acyclic Graph?
Or Distributed Architecture Guidepost!

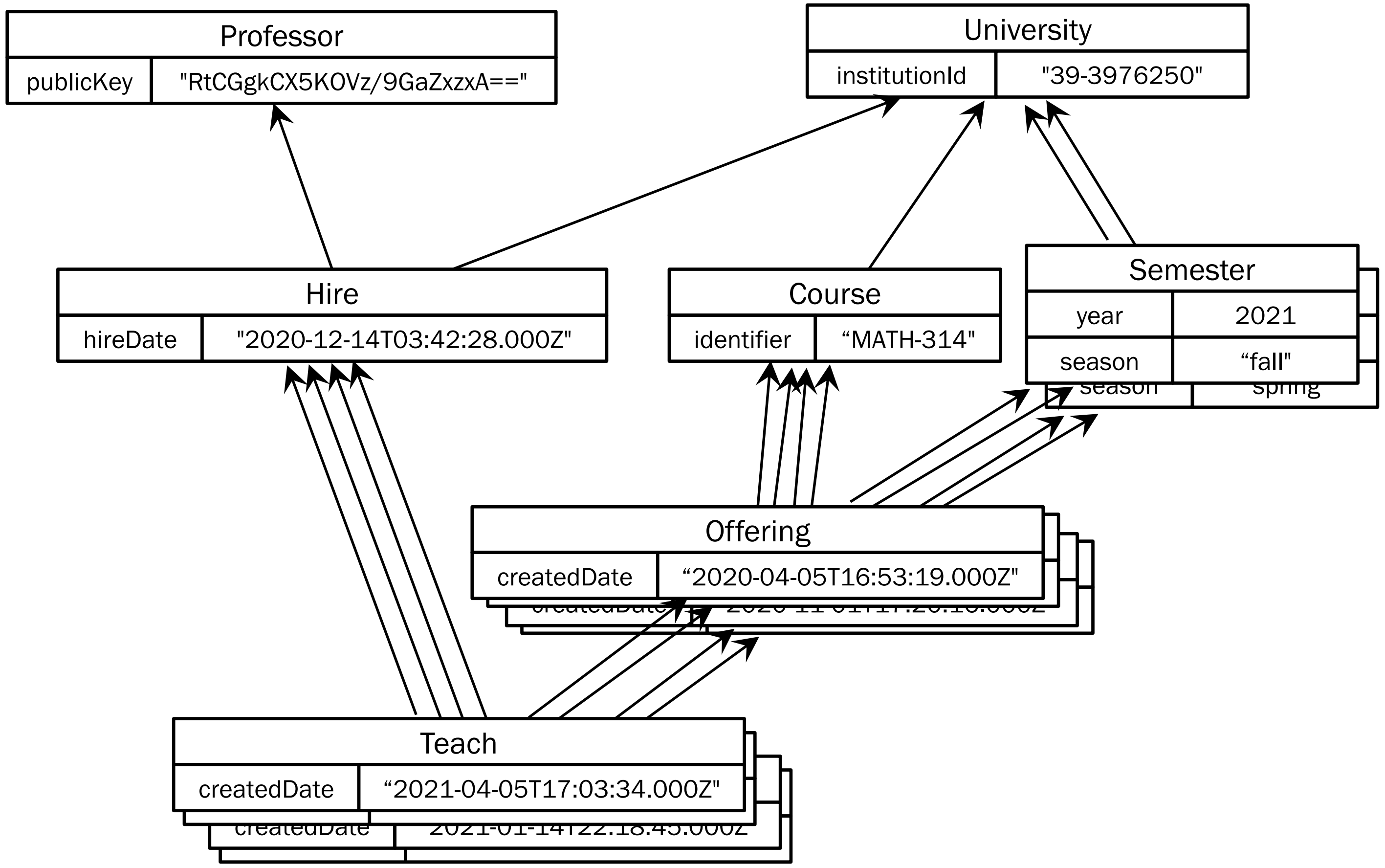
Causally related decisions, and
how they relate to application design

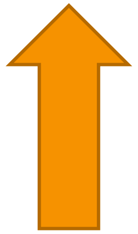
Michael L Perry
Improving


Professor
University
Student
Hire
Course
Semester
Enroll
Offering
Teach
Register
Drop
Fail
Complete


Professor
publicKey
"RtCGgkCX5KOVz/9GaZxzxA=="
University
institutionId
"39-3976250"
Hire
hireDate
"2020-12-14T03:42:28.000Z"
Course
identifier
“MATH-314"
Semester
year
2021
season
“spring"
Offering
createdDate
“2020-11-01T17:20:16.000Z"
Teach
createdDate
“2021-01-14T22:18:45.000Z"


University
institutionId
"39-3976250"
Student
publicKey
"nVYFqQfb4zpjoDInUKSjvQ=="
Course
identifier
"MATH-314"
Semester
year
2021
season
"spring"
Enroll
enrollmentDate
"2018-10-08T01:23:56.000Z"
Offering
createdDate
"2020-11-01T17:20:16.000Z"
Register
registrationDate
"2020-11-19T17:02:04.000Z"
Drop
Fail
grade
45.0
Complete
grade
87.2


Professor
publicKey
"RtCGgkCX5KOVz/9GaZxzxA=="
University
institutionId
"39-3976250"
Hire
hireDate
"2020-12-14T03:42:28.000Z"
Course
identifier
“MATH-314"
Semester
year
2021
season
“fall"
season
spring
Offering
createdDate
“2020-04-05T16:53:19.000Z"
createdDate
2020-11-01T17:20:18.000Z
Teach
createdDate
“2021-04-05T17:03:34.000Z"
createdDate
2021-01-14T22:18:45.000Z



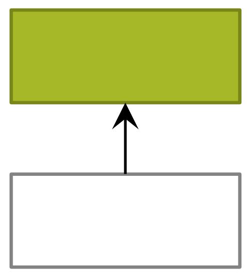
Causal Past

Directed Acyclic Graph

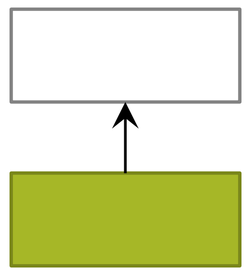
Each decision points back to its immediate causal predecessors

Fact

Predecessor

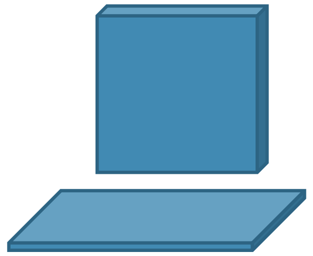


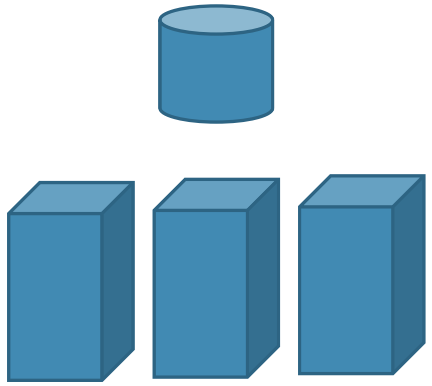
Successor

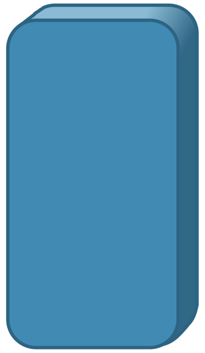


Immutable

University Software







Staff

Hire faculty

Faculty

Teach courses

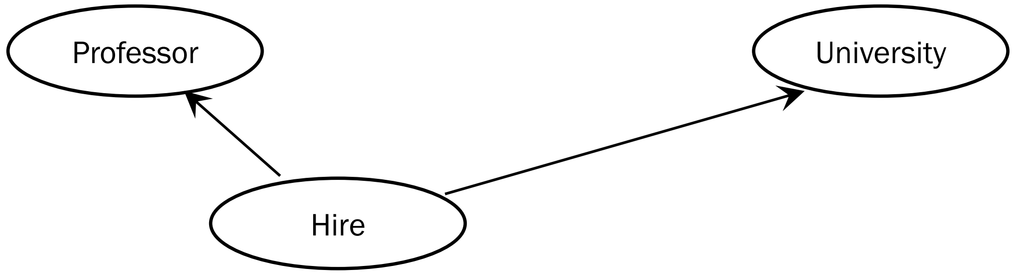
Students

Register for courses

Staff Application

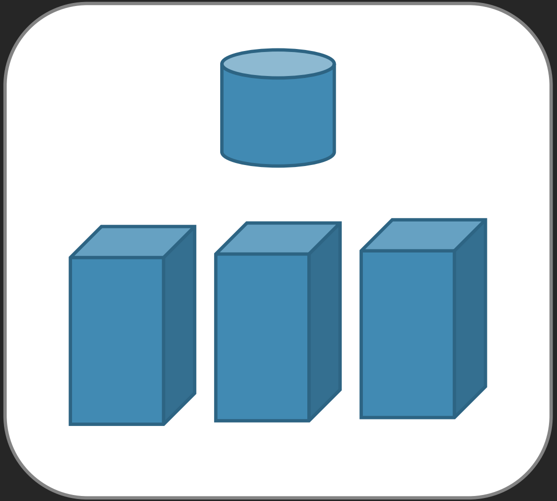
Desktop app for hiring faculty

Subset of types required for making hiring decisions

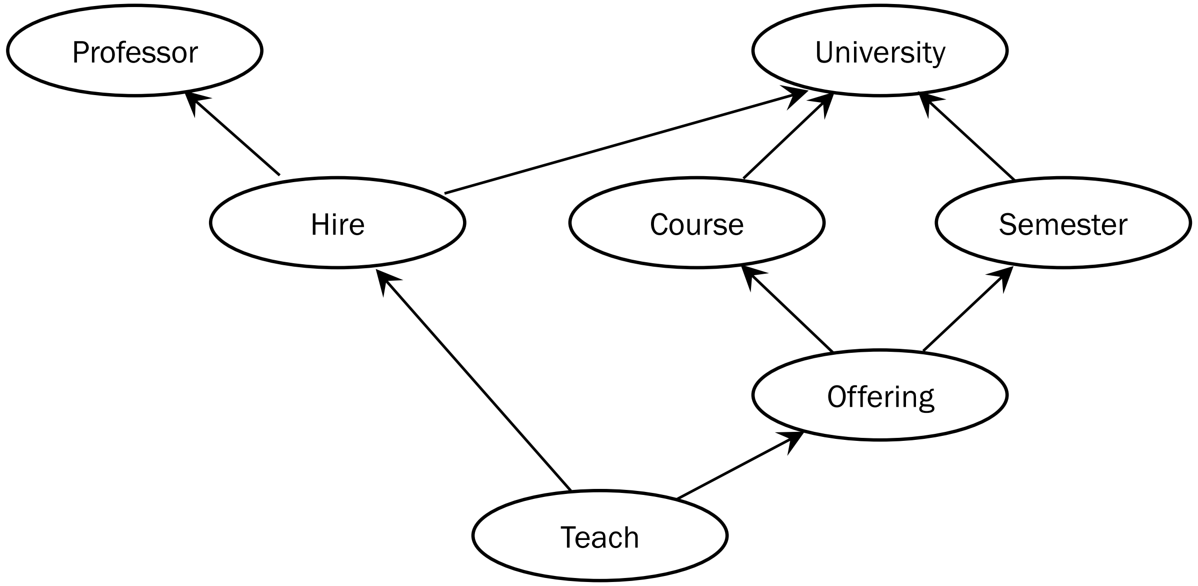


Faculty Application

Web app for managing course load



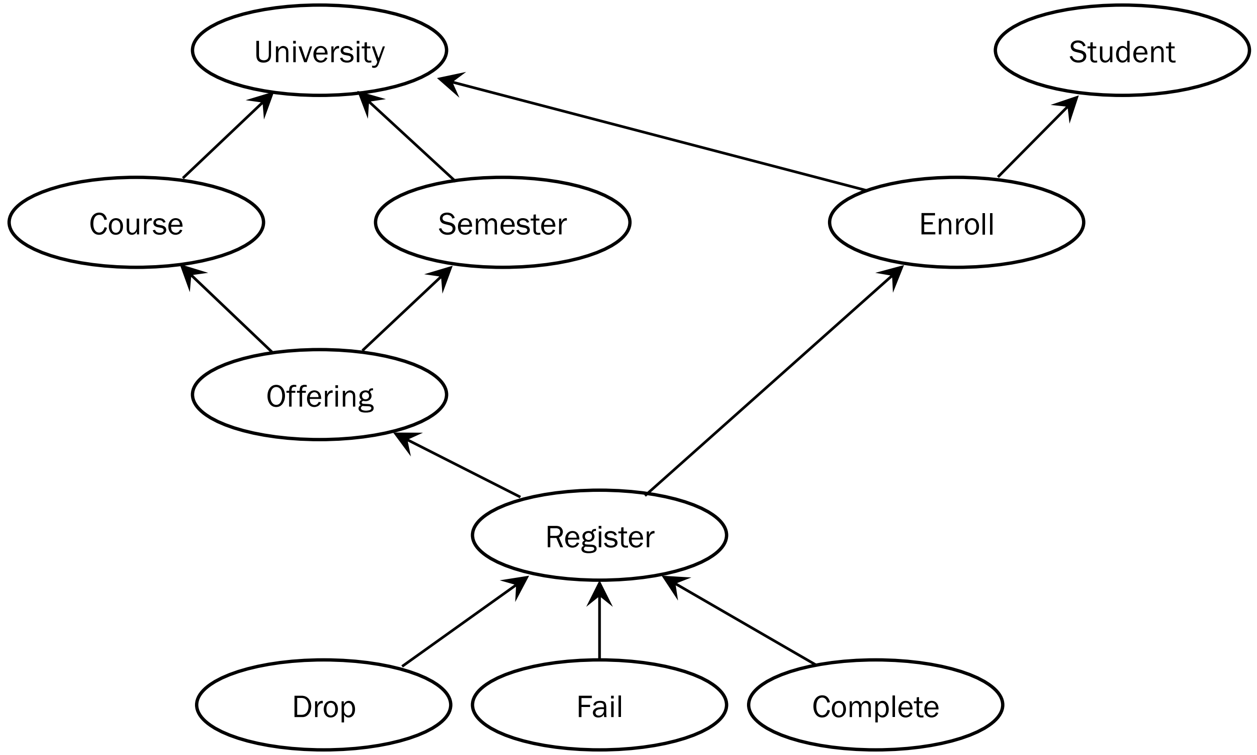
Subset of types required for managing course catalogs and offerings

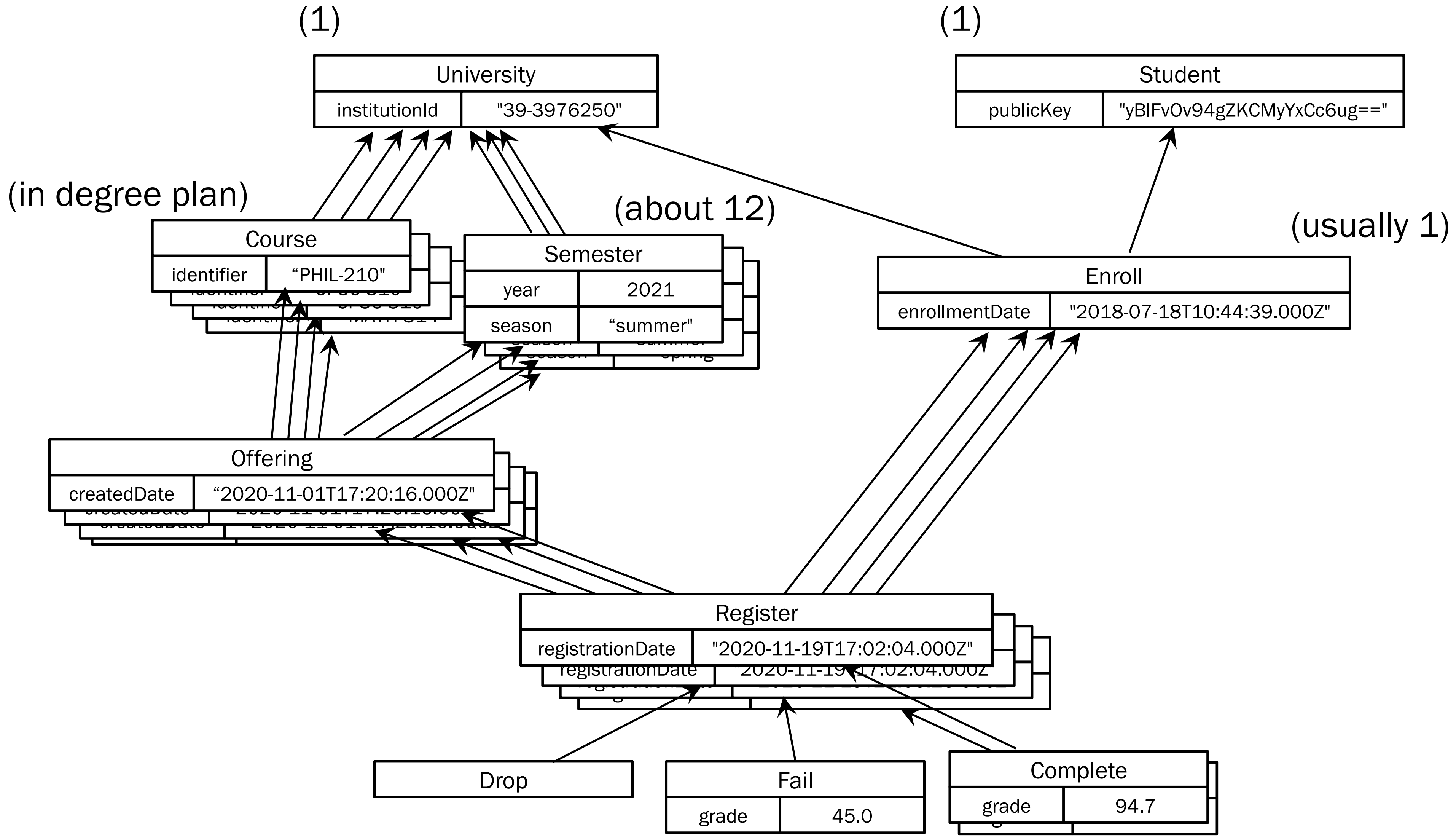


Student Application

Mobile app for course registration

Subset of types required for registering for courses




(1)
University
institutionId
"39-3976250"
(1)
Student
publicKey
"yBIFvOv94gZKCMyYxCc6ug=="
(in degree plan)
Course
identifier
“PHIL-210"
(about 12)
Semester
year
2021
season
“summer"
(usually 1)
Enroll
enrollmentDate
"2018-07-18T10:44:39.000Z"
Offering
createdDate
“2020-11-01T17:20:16.000Z"
Register
registrationDate
"2020-11-19T17:02:04.000Z"
Drop
Fail
grade
45.0
Complete
grade
94.7

SQL Table Design

Add a primary key.
All other columns are in the alternate key.

Define foreign keys for predecessors.
Foreign keys are in the alternate key too.

Deletion is not allowed.
Instead, insert a tombstone record.

<u>Professor</u>

PK	ProfessorId
AK	PublicKey

<u>University</u>

PK	UniversityId
AK	InstitutionId

<u>Hire</u>

PK	HireId
AK, FK	ProfessorId
AK, FK	UniversityId
AK	HireDate

<u>Terminate</u>

PK	TerminateId
AK, FK	HireId

Insert if Not Exists

Select the primary key using the alternate key.
If it is not found, insert using the alternate key.

```
SELECT ProfessorId
FROM Professor
WHERE PublicKey = $1
```

Iterate over this operation to merge DAGs.

If none

```
INSERT INTO Professor
  (PublicKey)
  VALUES (%1)
ON CONFLICT DO NOTHING
```

If most inserts will be new, reverse the order.

Query the DAG

Use an existential condition to filter out entities that should be deleted.

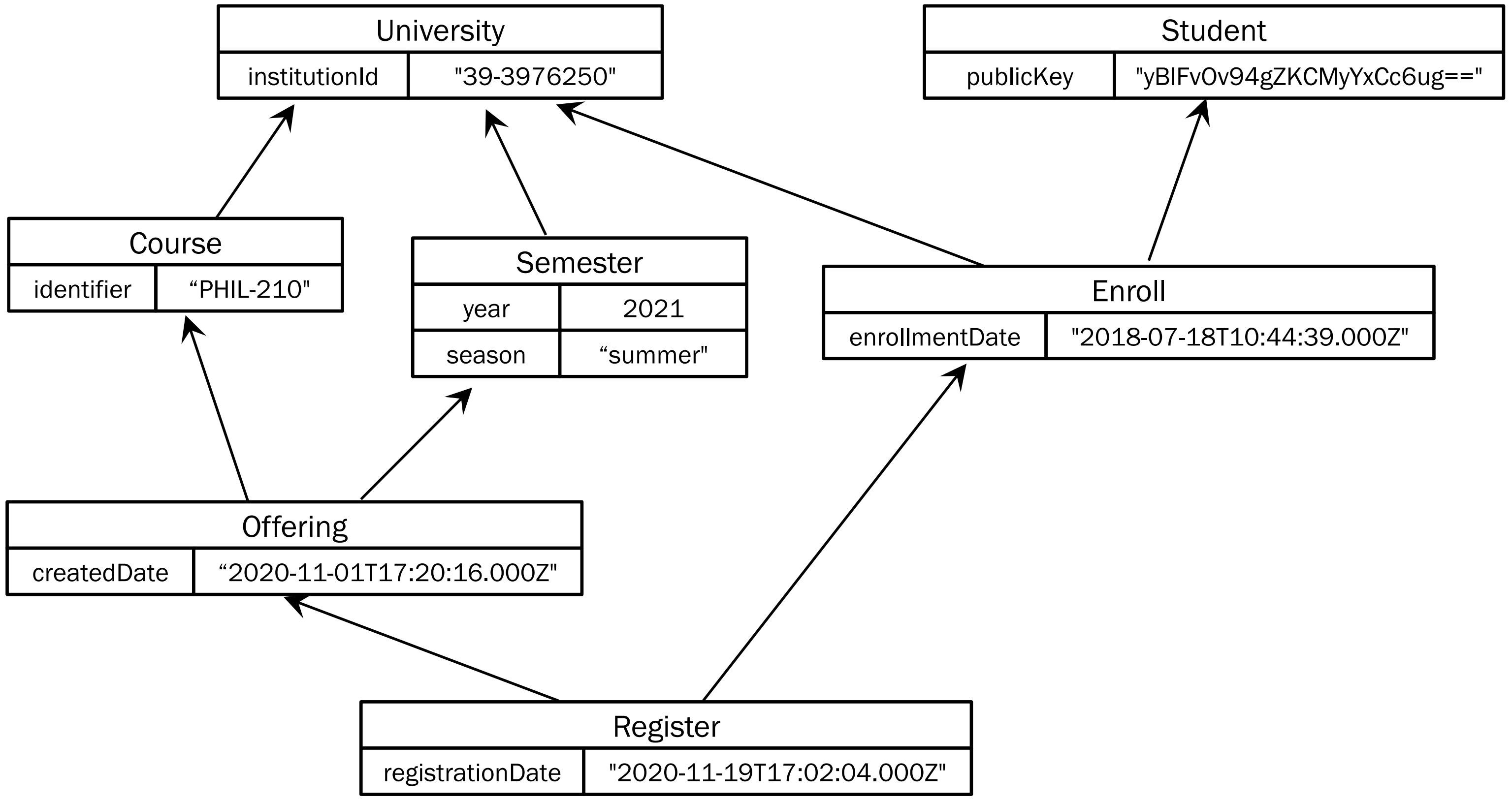
If most entities are eventually deleted, optimize with a materialized view.

```
SELECT p.ProfessorId
FROM Professor p
JOIN Hire h
  ON h.ProfessorId = p.ProfessorId
JOIN University u
  ON u.UniversityId = h.UniversityId
WHERE u.InstitutionId = %1
  AND NOT EXISTS (
    SELECT TerminateId
    FROM Terminate t
    WHERE t.HireId = h.HireId
  )
```

Transitive Closure

A subset that includes all predecessors of facts that are in the graph

Minimal set required to make sense of a fact



Message Design

A message contains the transitive closure of a fact.

Guarantees consistency when messages are delivered out of order.

Assumes that DAG can be correctly inferred from structures.

```json
{
   "register": {
      "registrationDate": "2021-11-19T17:02:04.000Z"
   },
   "offering": {
      "createdDate": "2020-11-01T17:20:16.000Z"
   },
   "course": {
      "identifier": "PHIL-210"
   },
   "semester": {
      "year": 2021,
      "season": "summer"
   },
   "enroll": {
      "enrollmentDate": "2018-07-18T10:44:39.000Z"
   },
   "university": {
      "institutionId": "39-3976250"
   },
   "student": {
      "publicKey": "yBlFvOv94gZKCMyYxCc6ug=="
   }
}
```

Merkle Tree

Compute the hash of a fact in canonical form.

```
$ echo -n '{"institutionId":"39-3976250"}' | openssl dgst -sha256 -binary | base64
sGoLdnSOB7dtTEabz8u8WxLthxbDVaIxSisWgIVdY4I=
```

Reference the hash in place of the predecessor.

```
{
  "enrollmentDate": "2018-07-18T10:44:39.000Z",
  "student": {
    "ref": "qvSCdGo6PjZ2mSG08xI6UFBO0HEbULXzvpZNW+WxVP8="
  },
  "university": {
    "ref": "sGoLdnSOB7dtTEabz8u8WxLthxbDVaIxSisWgIVdY4I="
  }
}
```

Merkle Tree

Include all facts by reference in the message.

```
{
  "sGoLdnSOB7dtTEabz8u8WxLthxbDVaIxSisWgIVdY4I=": {
    "institutionId":"39-3976250"
  },
  "qvSCdGo6PjZ2mSG08xI6UFBO0HEbULXzvpZNW+WxVP8=": {
    "publicKey": "yBlFvOv94gZKCMyYxCc6ug=="
  },
  "RXNLfyirFx1BMHXqGMc/SzyVGaN6Qul2hIPCuP0jgfk=": {
    "enrollmentDate": "2018-07-18T10:44:39.000Z",
    "student": {
      "ref": "qvSCdGo6PjZ2mSG08xI6UFBO0HEbULXzvpZNW+WxVP8="
    },
    "university": {
      "ref": "sGoLdnSOB7dtTEabz8u8WxLthxbDVaIxSisWgIVdY4I="
    }
  }
}
```

Eventual Consistency

Ensure that all nodes reach the same state after receiving the same information

Idempotent

- Receiving a message twice does not duplicate the effect.
- Recognize duplicate messages by their hash.
- “Insert if not exists” will prevent duplicates.

Commutative

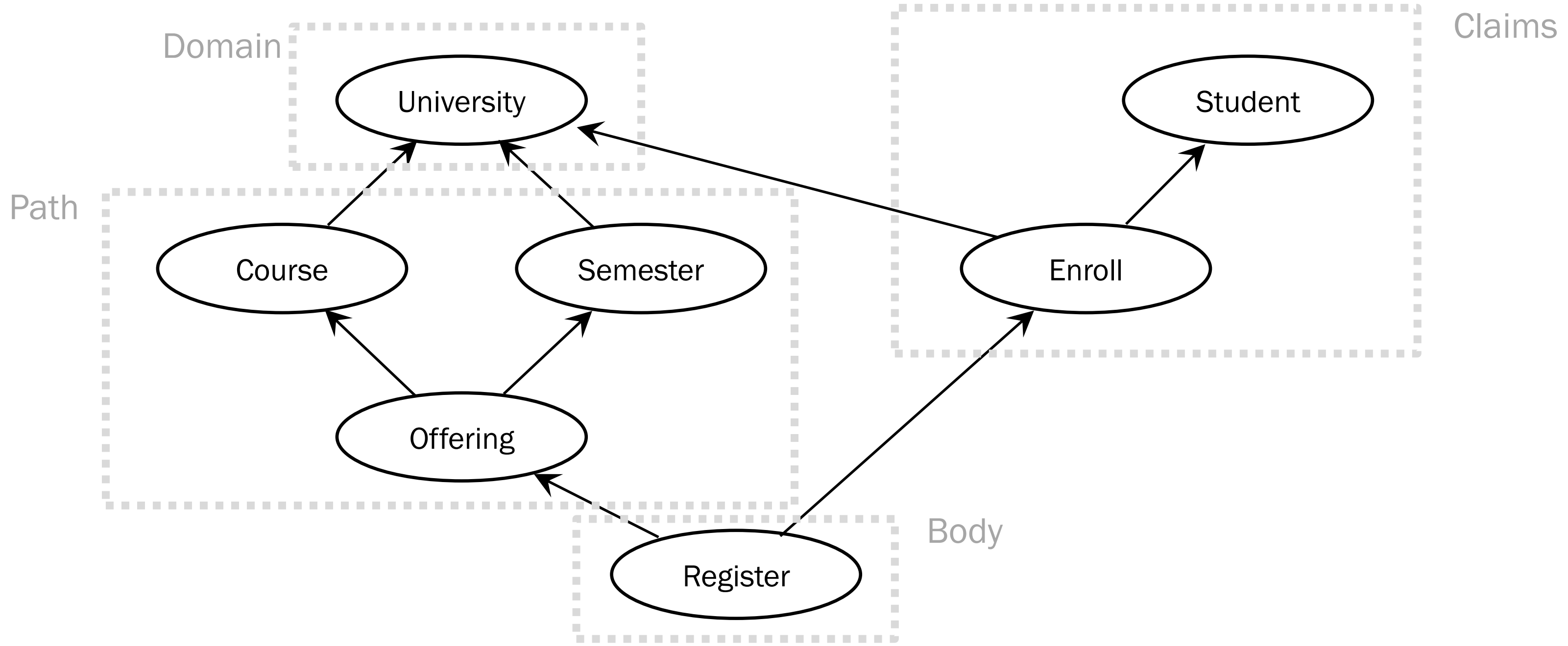
- Messages contain their predecessors.
- If order matters, then the earlier message is delivered with the later one.
- If there is no causal relationship, then order doesn’t matter.

Associative

- A merge between two DAGs computes the least upper bound.
- This is a general-purpose CRDT.

API Design

Every API request includes the transitive closure.
Different parts of the graph are stored in different parts of the request.



API Design

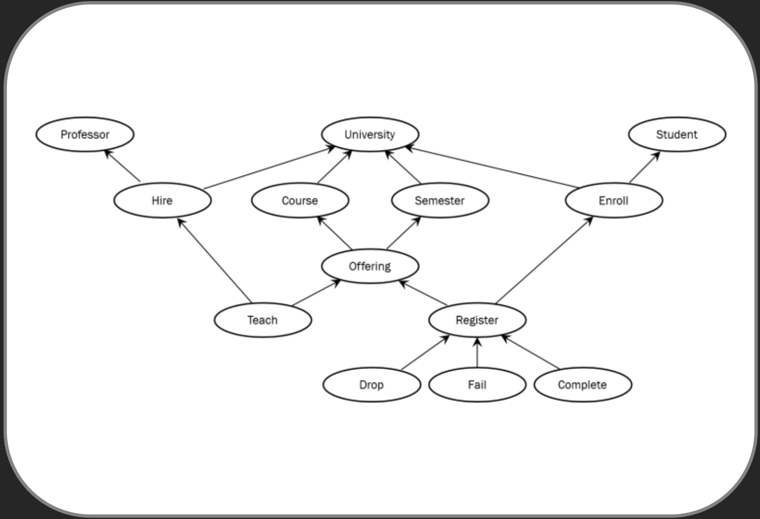
```
POST fairviewcollege.edu/courses/2021/summer/PHIL-210/2020-11-01T17:20:16.000Z

{
  "registrationDate": "2020-11-19T17:02:04.000Z"
}
```

```
claims:
"publicKey": "yBlFvOv94gZKCMyYxCc6ug==",
"enrollmentDate": "2018-07-18T10:44:39.000Z"
```

The registration identifier is generated on the client side, where the decision is made.
This guarantees that registration is idempotent.

Directed Acyclic Graphs

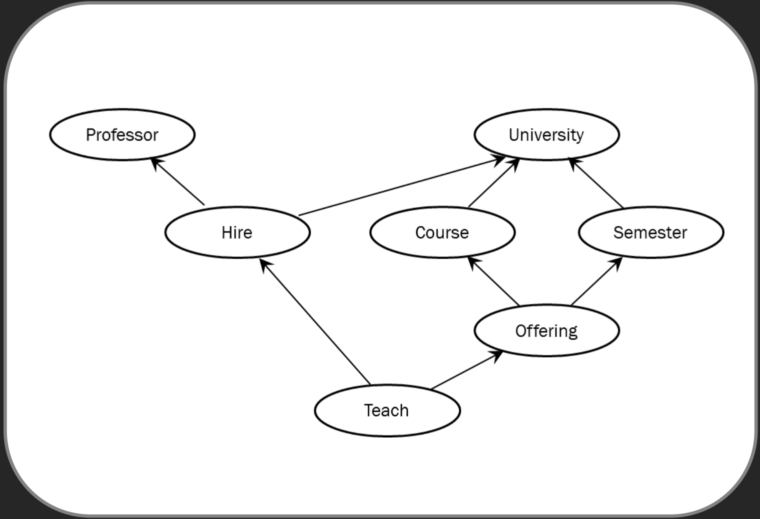
The key to domain modeling and distributed systems design



Model a problem domain as a set of causally related decisions.

Directed Acyclic Graphs

The key to domain modeling and distributed systems design

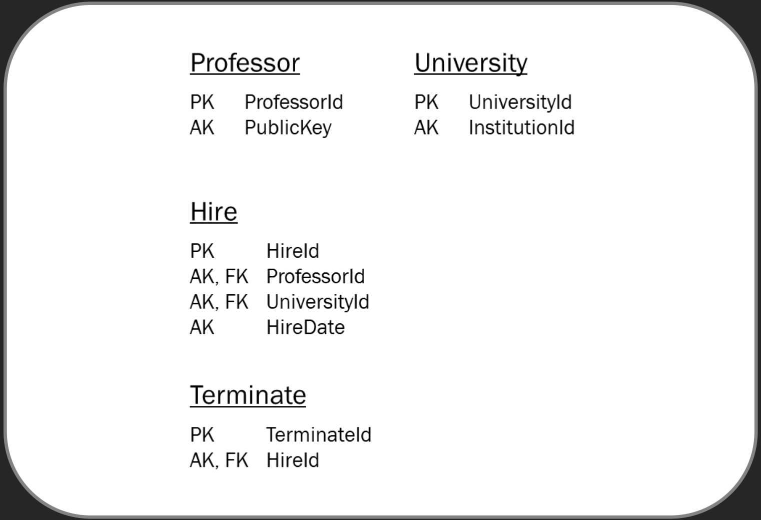


Model a problem domain as a set of causally related decisions.

Identify subsets for different applications, processes, and microservices.

Directed Acyclic Graphs

The key to domain modeling and distributed systems design



Professor

PK	ProfessorId
AK	PublicKey

University

PK	UniversityId
AK	InstitutionId

Hire

PK	HireId
AK, FK	ProfessorId
AK, FK	UniversityId
AK	HireDate

Terminate

PK	TerminateId
AK, FK	HireId

Model a problem domain as a set of causally related decisions.

Identify subsets for different applications, processes, and microservices.

Create a table per fact type, using foreign keys to refer to predecessors.

Directed Acyclic Graphs

The key to domain modeling and distributed systems design

```
SELECT ProfessorId
FROM Professor
WHERE PublicKey = $1
```

If none

```
INSERT INTO Professor
  (PublicKey)
  VALUES (%1)
ON CONFLICT DO NOTHING
```

Model a problem domain as a set of causally related decisions.

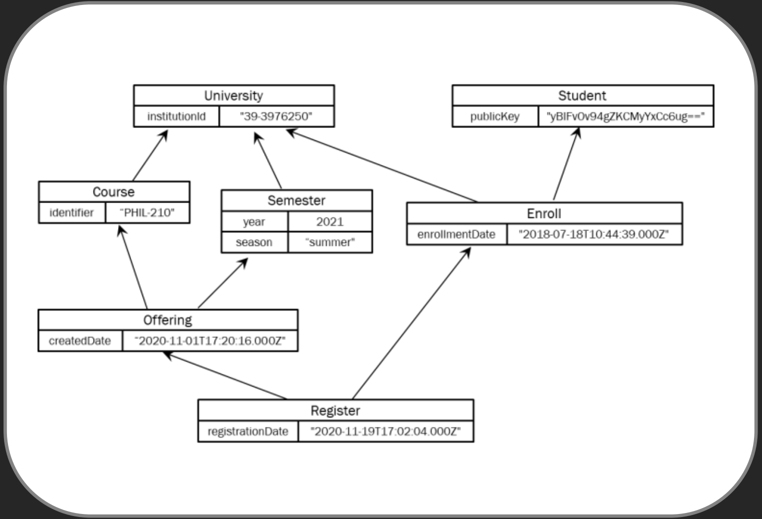
Identify subsets for different applications, processes, and microservices.

Create a table per fact type, using foreign keys to refer to predecessors.

Insert if not exists to merge into the DAG.

Directed Acyclic Graphs

The key to domain modeling and distributed systems design



Model a problem domain as a set of causally related decisions.

Identify subsets for different applications, processes, and microservices.

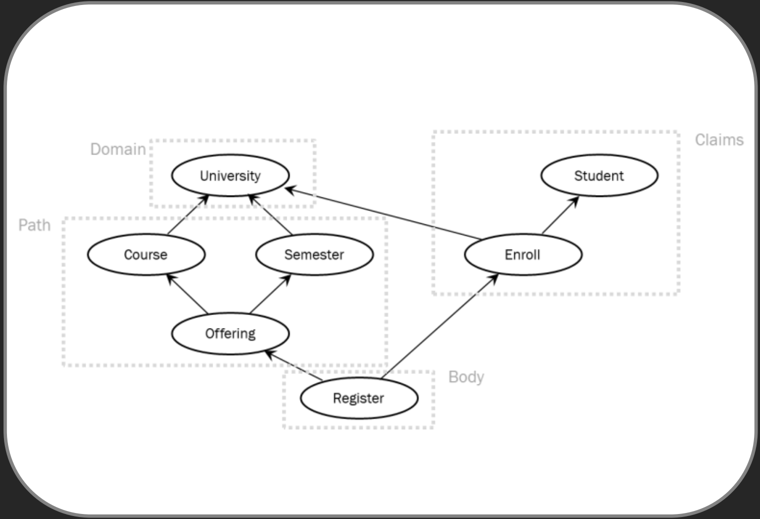
Create a table per fact type, using foreign keys to refer to predecessors.

Insert if not exists to merge into the DAG.

Compute the transitive closure to determine the information that should be included in a message.

Directed Acyclic Graphs

The key to domain modeling and distributed systems design



Model a problem domain as a set of causally related decisions.

Identify subsets for different applications, processes, and microservices.

Create a table per fact type, using foreign keys to refer to predecessors.

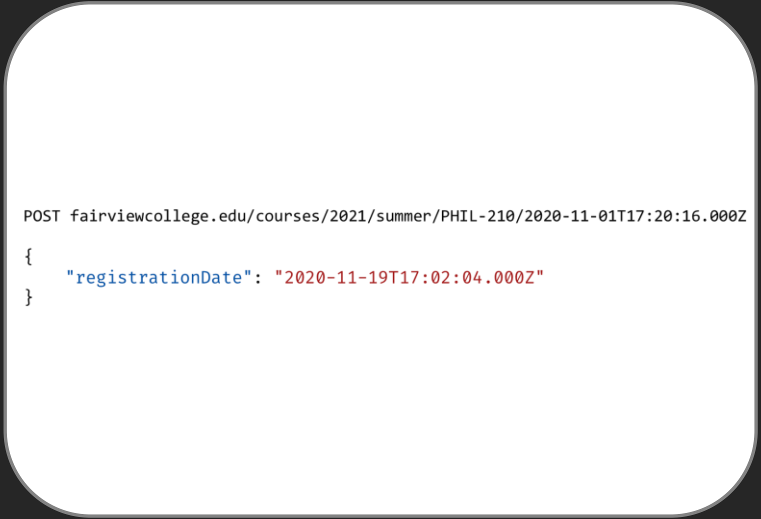
Insert if not exists to merge into the DAG.

Compute the transitive closure to determine the information that should be included in a message.

Allocate different parts of the transitive closure to different parts of each API request.

Directed Acyclic Graphs

The key to domain modeling and distributed systems design



```
POST fairviewcollege.edu/courses/2021/summer/PHIL-210/2020-11-01T17:20:16.000Z

{
    "registrationDate": "2020-11-19T17:02:04.000Z"
}
```

Model a problem domain as a set of causally related decisions.

Identify subsets for different applications, processes, and microservices.

Create a table per fact type, using foreign keys to refer to predecessors.

Insert if not exists to merge into the DAG.

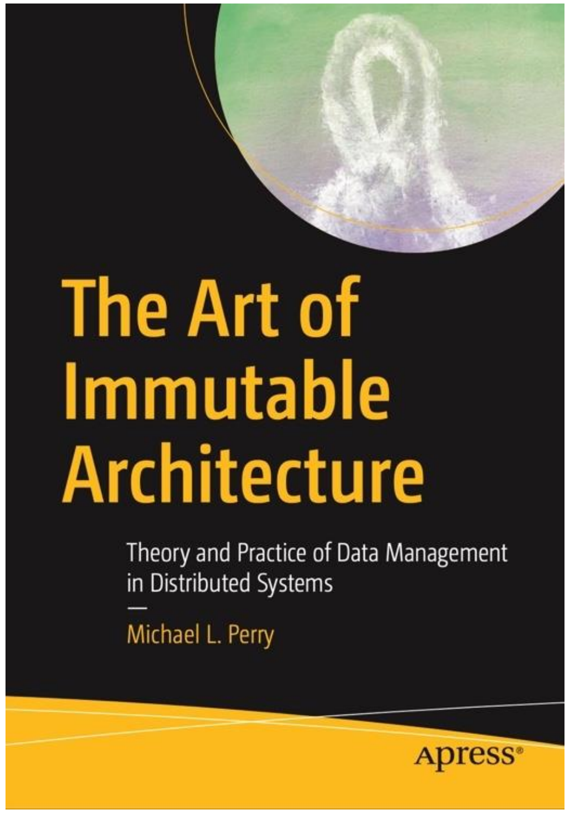
Compute the transitive closure to determine the information that should be included in a message.

Allocate different parts of the transitive closure to different parts of each API request.

Generate identifiers where decisions are made.

Learn More

ImmutableArchitecture.com
@MichaelLPerry
Michael.Perry@Improving.com


The Art of Immutable Architecture
Theory and Practice of Data Management in Distributed Systems
Michael L. Perry
Apress