



Practical object-oriented models in SQL

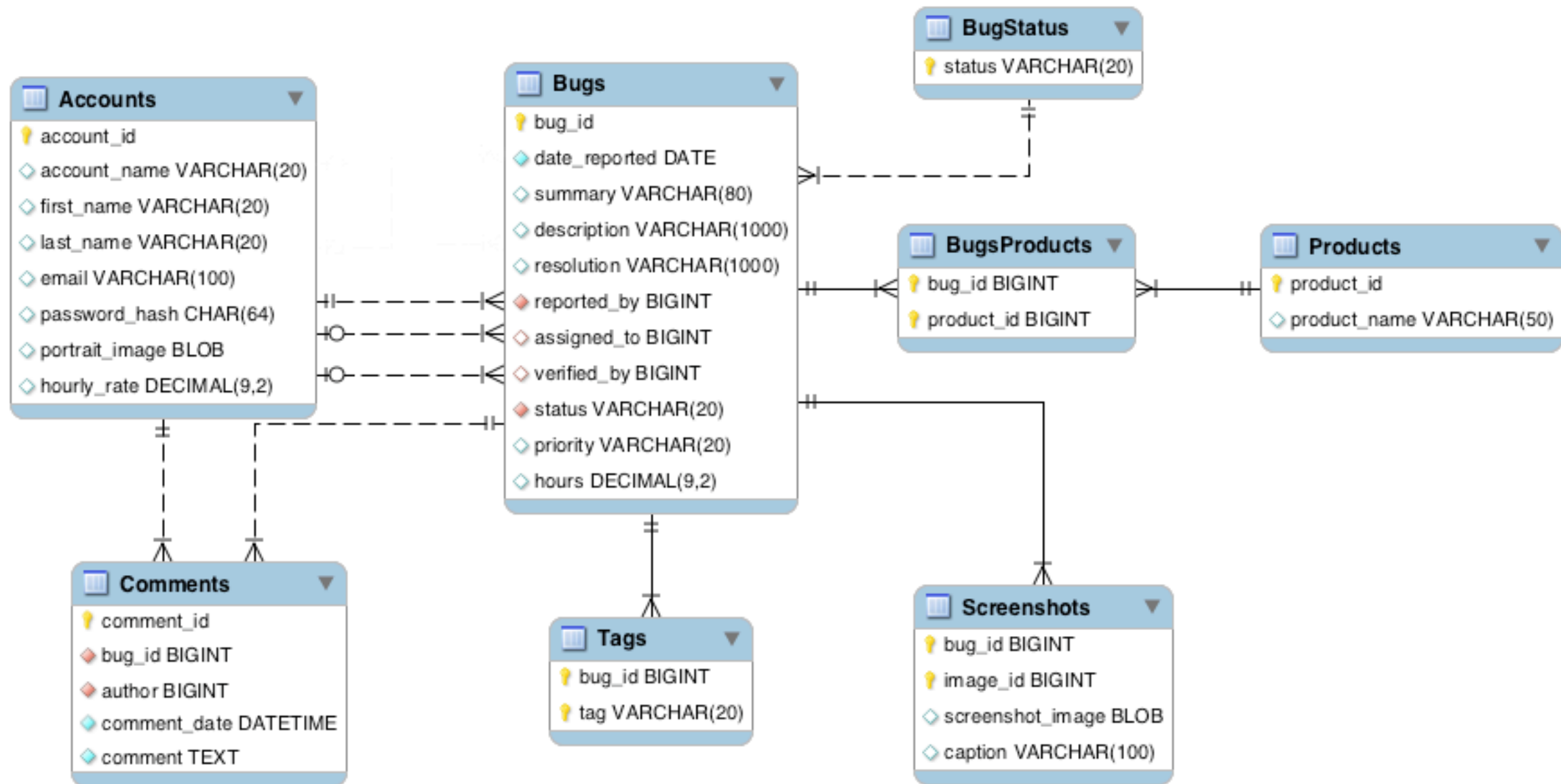
Bill Karwin
July 22, 2009 - OSCON

Object-Oriented vs. Relational

- Impedance mismatch
- OO operates on instances; RDBMS operates on sets
- Compromise either OO strengths or relational strengths



Example database



Entity-Attribute-Value

If you try and take a cat apart to see how it works, the first thing you have on your hands is a non-working cat.

— Richard Dawkins

Entity-Attribute-Value

- **Objective:** extensibility
 - Variable sets of attributes

bug_id	bug_type	priority	description	severity	sponsor
1234	BUG	high	crashes when saving	loss of functionality	
3456	FEATURE	low	support XML		Acme Corp.

Entity-Attribute-Value Meta-Table

```
CREATE TABLE eav (  
    bug_id      BIGINT NOT NULL,  
    attr_name   VARCHAR(20) NOT NULL,  
    attr_value  VARCHAR(100),  
    PRIMARY KEY (bug_id, attr_name),  
    FOREIGN KEY (bug_id) REFERENCES Bugs(bug_id)  
);
```

*mixing data
with metadata*



What does this look like?

bug_id	attr_name	attr_value
1234	priority	high
1234	description	crashes when saving
1234	severity	loss of functionality
3456	priority	low
3456	description	support XML
3456	sponsor	Acme Corp.

Problems: names

bug_id	attr_name	attr_value
1234	created	2008-04-01
3456	created_date	2008-04-01

Entity-Attribute-Value

Problems: values

bug_id	attr_name	attr_value
1234	created_date	2008-02-31
3456	created_date	banana

Problems: constraints

NOT NULL

- Difficult to enforce mandatory attributes
 - SQL constraints apply to columns, not rows
 - No way to declare that a row must exist with a certain *attr_name* value ('created_date')

Problems: constraints

- Difficult to use lookup tables

bug_id	attr_name	attr_value
1234	priority	new
3456	priority	fixed
5678	priority	banana

- Constraints apply to all rows in the column, not selected rows depending on value in *attr_name*

Problems: queries

- Difficult to reconstruct a row of attributes:

```
SELECT b.bug_id,  
       e1.attr_value AS created_date,  
       e2.attr_value AS priority,  
       e3.attr_value AS description,  
       e4.attr_value AS status,  
       e5.attr_value AS reported_by
```

```
FROM Bugs b
```

```
LEFT JOIN eav e1 ON (b.bug_id = e1.bug_id AND e1.attr_name = 'created_date')
```


```
LEFT JOIN eav e2 ON (b.bug_id = e2.bug_id AND e2.attr_name = 'priority')
```

```
LEFT JOIN eav e3 ON (b.bug_id = e3.bug_id AND e3.attr_name = 'description')
```

```
LEFT JOIN eav e4 ON (b.bug_id = e4.bug_id AND e4.attr_name = 'status')
```

```
LEFT JOIN eav e5 ON (b.bug_id = e5.bug_id AND e5.attr_name = 'reported_by');
```

*need one JOIN
per attribute*



bug_id	created_date	priority	description	status	reported_by
1234	2008-04-01	high	Crashes when I save.	NEW	Bill

Entity-Attribute-Value

Solution?

Do it all in application logic?

Entity-Attribute-Value

Solution

Use *metadata* for metadata

- Define attributes in columns
- ALTER TABLE to add attribute columns
- Define related tables for related types

Single Table Inheritance

- One table with many columns
- Inapplicable columns are NULL

```
CREATE TABLE Issues (  
    issue_id        SERIAL PRIMARY KEY,  
    created_date    DATE NOT NULL,  
    priority        VARCHAR(20),  
    description     TEXT,  
    issue_type      CHAR(1) CHECK (issue_type IN ('B','F')),  
    bug_severity    VARCHAR(20),  
    feature_sponsor VARCHAR(100)  
);
```


Concrete Table Inheritance

- Define similar tables for similar types
- Duplicate common columns in each table

```
CREATE TABLE Bugs (  
    bug_id      SERIAL PRIMARY KEY,  
    created_date DATE NOT NULL,  
    priority    VARCHAR(20),  
    description TEXT,  
    severity    VARCHAR(20)  
);
```

```
CREATE TABLE Features (  
    bug_id      SERIAL PRIMARY KEY,  
    created_date DATE NOT NULL,  
    priority    VARCHAR(20),  
    description TEXT,  
    sponsor    VARCHAR(100)  
);
```


Concrete Table Inheritance

- Use UNION to search both tables:

```
SELECT u.* FROM (  
    SELECT issue_id, description FROM Bugs  
    UNION ALL  
    SELECT issue_id, description FROM Features  
) u  
WHERE u.description LIKE ...
```


Class Table Inheritance

- Common columns in base table
- Subtype-specific columns in subtype tables

```
CREATE TABLE Bugs (  
  issue_id BIGINT PRIMARY KEY,  
  severity VARCHAR(20),  
  FOREIGN KEY (issue_id)  
    REFERENCES Issues (issue_id)  
);
```

```
CREATE TABLE Features (  
  issue_id BIGINT PRIMARY KEY,  
  sponsor VARCHAR(100),  
  FOREIGN KEY (issue_id)  
    REFERENCES Issues (issue_id)  
);
```

```
CREATE TABLE Issues (  
  issue_id SERIAL PRIMARY KEY,  
  created_date DATE NOT NULL  
  priority VARCHAR(20),  
  description TEXT  
);
```


Class Table Inheritance

- Easy to query common columns:

```
SELECT * FROM Issues  
WHERE description LIKE ...;
```

- Easy to query one subtype at a time:

```
SELECT * FROM Issues  
JOIN Bugs USING (issue_id)  
WHERE description LIKE ...;
```


Using EAV appropriately

- If attributes must be fully dynamic
- Enforce constraints in application code
- Don't try to fetch a single row per entity
- Consider non-relational solutions for semi-structured data, e.g. RDF/XML

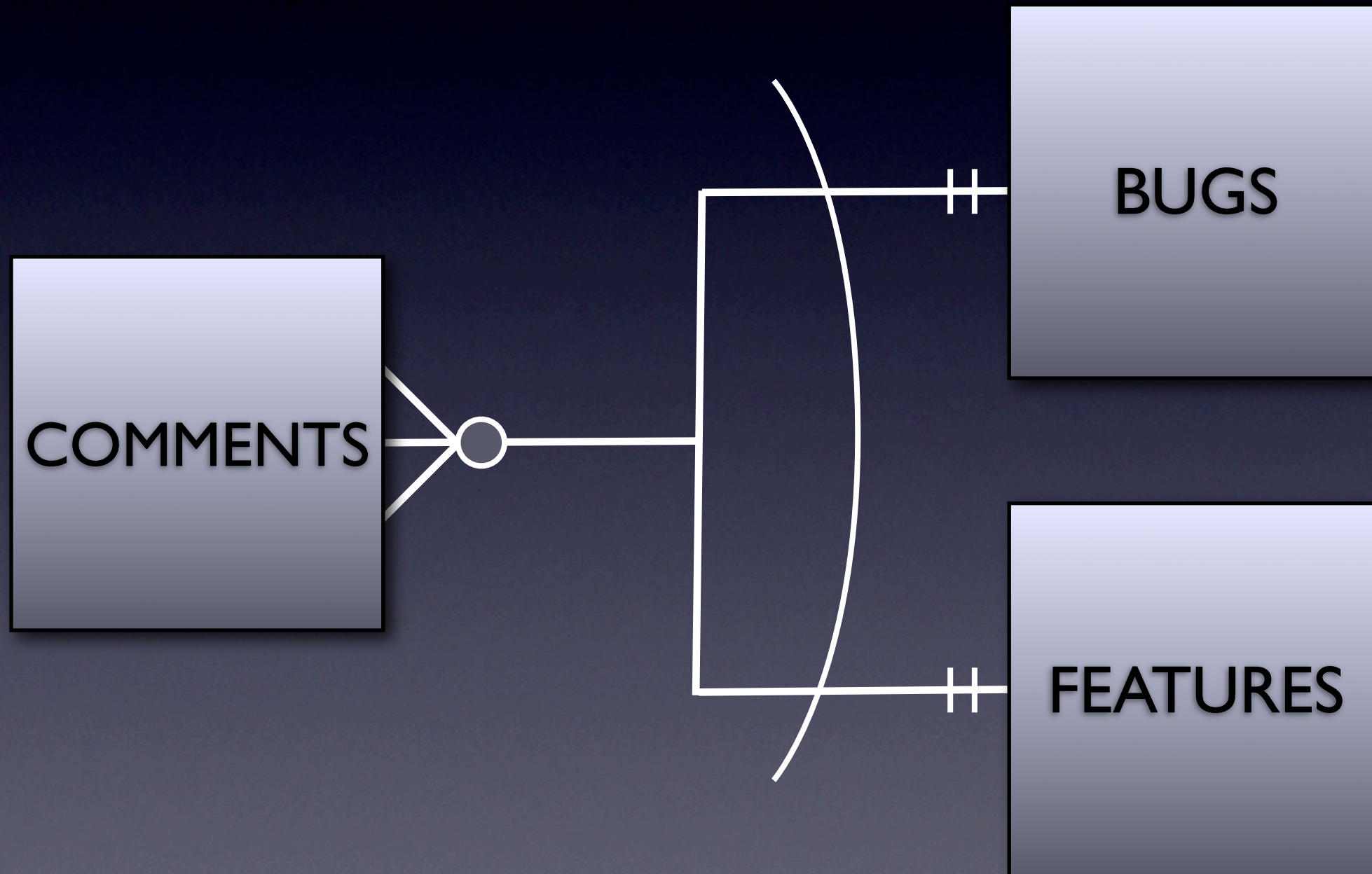
Polymorphic Associations

Of course, some people do go both ways.

— The Scarecrow

Polymorphic Associations

- **Objective:** reference multiple parents



What does this look like?


mixing data with metadata

Query result:

comment_id	comment	issue_type	c.issue_id	b.issue_id	f.issue_id
6789	“It crashes”	Bug	1234	1234	NULL
9876	“Great idea!”	Feature	2345	NULL	2345

Problem: constraints

- A FOREIGN KEY constraint can't reference two tables:

```
CREATE TABLE Comments (  
  comment_id SERIAL PRIMARY KEY,  
  comment TEXT NOT NULL,  
  issue_type VARCHAR(15) NOT NULL  
    CHECK (issue_type IN ('Bugs', 'Features')),  
  issue_id BIGINT NOT NULL,  
  FOREIGN KEY issue_id REFERENCES   
);
```

*you need this to be
Bugs or Features*



Problem: constraints

- You have to define table with no referential integrity:

```
CREATE TABLE Comments (  
    comment_id SERIAL PRIMARY KEY,  
    comment TEXT NOT NULL,  
    issue_type VARCHAR(15) NOT NULL  
        CHECK (issue_type IN ('Bugs', 'Features')),  
    issue_id BIGINT NOT NULL  
);
```


Problem: queries

- You can't use a different table for each row.

```
SELECT * FROM Comments  
JOIN [redacted] USING (issue_id);
```


*you need this to be
Bugs or Features*



Problem: queries

- You have to join to each parent table:

```
SELECT *  
FROM Comments c  
LEFT JOIN Bugs b ON (c.issue_type = 'Bugs'  
AND c.issue_id = b.issue_id)  
LEFT JOIN Features f ON (c.issue_type = 'Features'  
AND c.issue_id = f.issue_id);
```



*you have to get
these strings right*

Polymorphic Associations

Solutions

- Exclusive arcs
- Reverse the relationship
- Base parent table

Exclusive Arcs

```
CREATE TABLE Comments (  
  comment_id SERIAL PRIMARY KEY,  
  comment TEXT NOT NULL,  
  bug_id BIGINT,  
  feature_id BIGINT,  
  FOREIGN KEY bug_id  
    REFERENCES Bugs(bug_id),  
  FOREIGN KEY feature_id  
    REFERENCES Features(feature_id)  
);
```

*both columns nullable;
exactly one must be non-null*

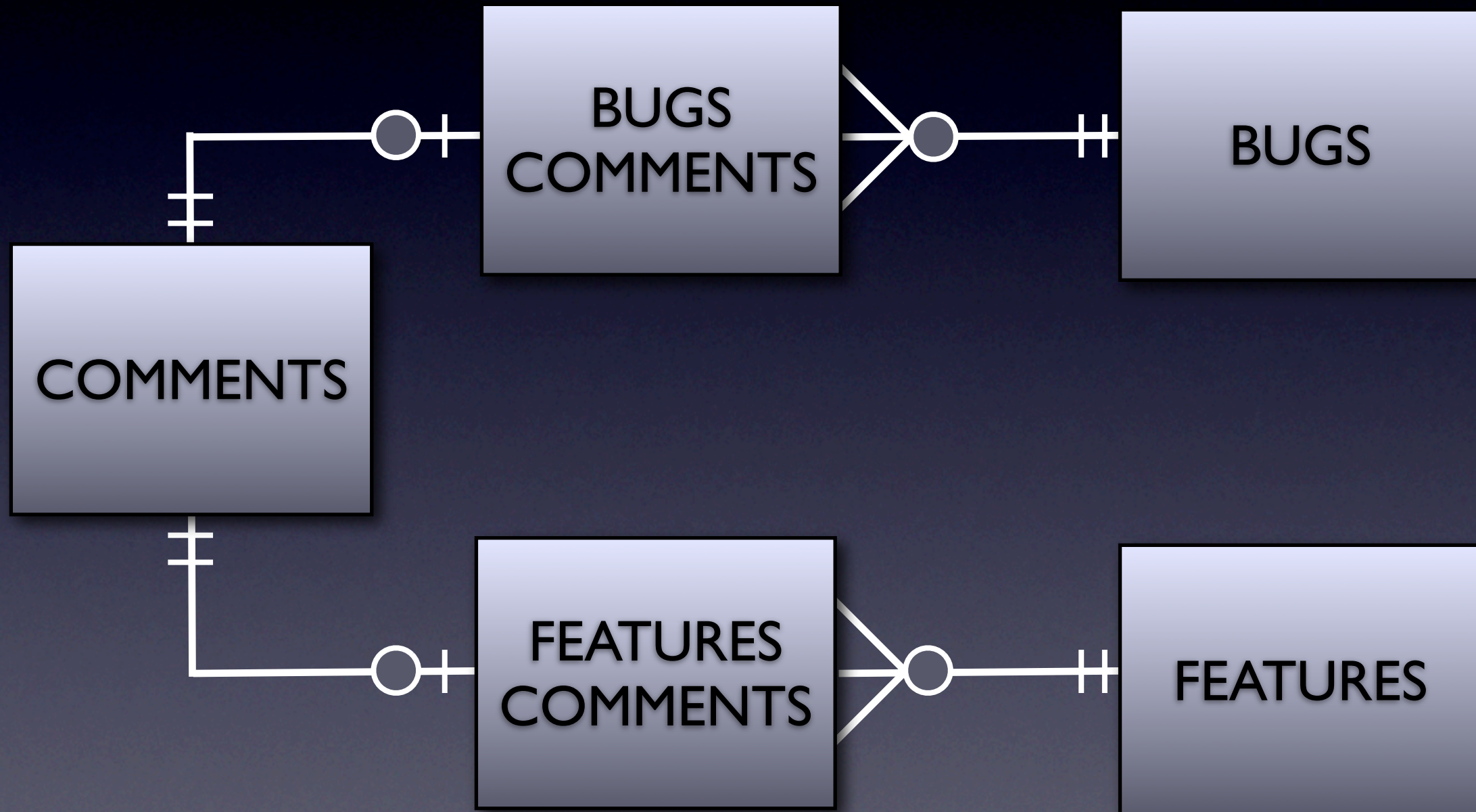


Exclusive Arcs

- Referential integrity is enforced
- But hard to ensure exactly one is non-null
- Queries are easier:

```
SELECT * FROM Comments c  
LEFT JOIN Bugs b USING (bug_id)  
LEFT JOIN Features f USING (feature_id);
```



Reverse the relationship



Reverse the relationship

```
CREATE TABLE BugsComments (  
  comment_id BIGINT NOT NULL,  
  bug_id BIGINT NOT NULL,  
  PRIMARY KEY (comment_id),  
  FOREIGN KEY (comment_id)  
    REFERENCES Comments(comment_id),  
  FOREIGN KEY (bug_id) REFERENCES Bugs(bug_id)  
);
```

not many-to-many



```
CREATE TABLE FeaturesComments (  
  comment_id BIGINT NOT NULL,  
  feature_id BIGINT NOT NULL,  
  PRIMARY KEY (comment_id),  
  FOREIGN KEY (comment_id)  
    REFERENCES Comments(comment_id),  
  FOREIGN KEY (feature_id) REFERENCES Features(feature_id)  
);
```


Reverse the relationship

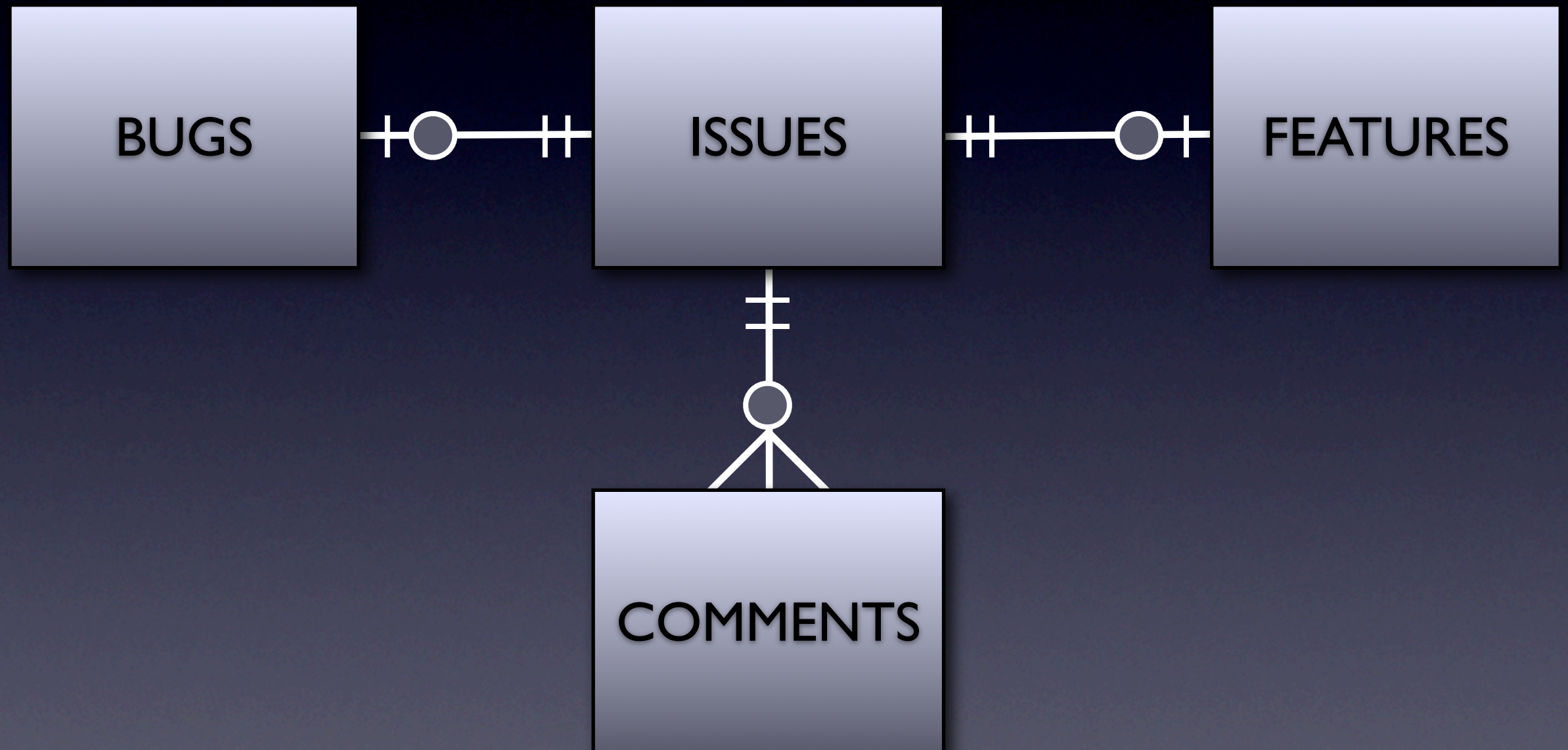
- Referential integrity is enforced
- Query comments for a given bug:

```
SELECT * FROM BugsComments b  
JOIN Comments c USING (comment_id)  
WHERE b.bug_id = 1234;
```

- Query bug/feature for a given comment:

```
SELECT * FROM Comments  
LEFT JOIN (BugsComments JOIN Bugs USING (bug_id))  
    USING (comment_id)  
LEFT JOIN (FeaturesComments JOIN Features USING (feature_id))  
    USING (comment_id)  
WHERE comment_id = 9876;
```


Base parent table



Base parent table

```
CREATE TABLE Issues (  
  issue_id SERIAL PRIMARY KEY  
);
```

```
CREATE TABLE Bugs (  
  issue_id BIGINT PRIMARY KEY,  
  ...  
  FOREIGN KEY (issue_id) REFERENCES Issues(issue_id)  
);
```

```
CREATE TABLE Comments (  
  comment_id SERIAL PRIMARY KEY,  
  comment TEXT NOT NULL,  
  issue_id BIGINT NOT NULL,  
  FOREIGN KEY (issue_id) REFERENCES Issues(issue_id)  
);
```

*works great with
Class Table
Inheritance*


Base parent table

- Referential integrity is enforced
- Queries are easier:

```
SELECT * FROM Comments  
JOIN Issues USING (issue_id)  
LEFT JOIN Bugs USING (issue_id)  
LEFT JOIN Features USING (issue_id);
```


Enforcing disjoint subtypes

```
CREATE TABLE Issues (  
    issue_id    SERIAL PRIMARY KEY,  
    issue_type  CHAR(1) NOT NULL  
                CHECK (issue_type IN ('B', 'F')),  
    UNIQUE KEY (issue_id, issue_type)  
);  
  
CREATE TABLE Bugs (  
    issue_id    BIGINT PRIMARY KEY,  
    issue_type  CHAR(1) NOT NULL  
                CHECK (issue_type = 'B'),  
    ...  
    FOREIGN KEY (issue_id, issue_type)  
        REFERENCES Issues(issue_id, issue_type)  
);
```



*referential
integrity*

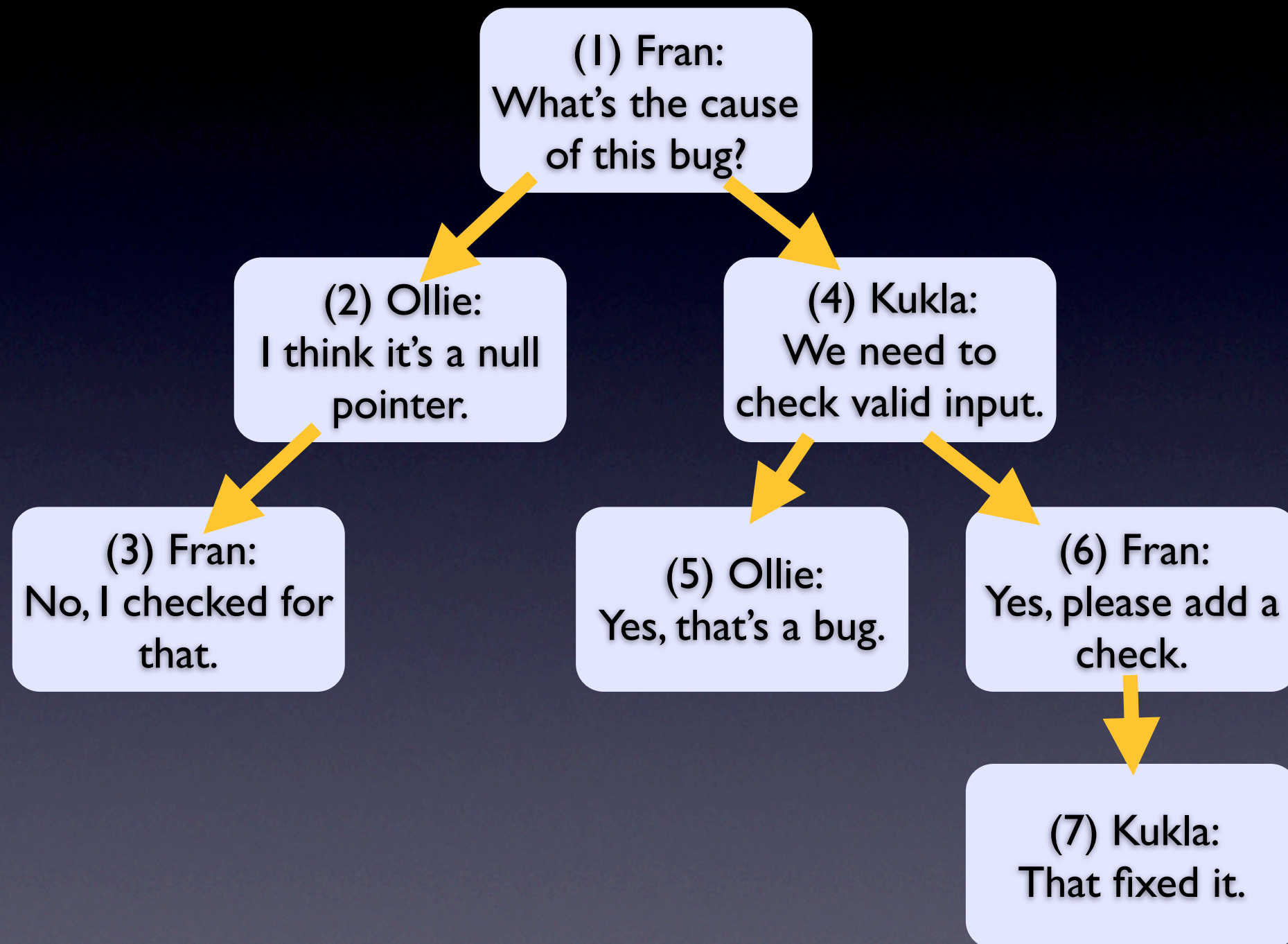
Naive Trees

A tree is a tree — how many more do you need to look at?
— Ronald Reagan

Naive Trees

- **Objective:** store & query hierarchical data
 - Categories/subcategories
 - Bill of materials
 - Threaded discussions

What does this look like?



Adjacency List design

- Naive solution nearly everyone uses
- Each entry knows its immediate parent

comment_id	parent_id	author	comment
1	NULL	Fran	What's the cause of this bug?
2	1	Ollie	I think it's a null pointer.
3	2	Fran	No, I checked for that.
4	1	Kukla	We need to check valid input.
5	4	Ollie	Yes, that's a bug.
6	4	Fran	Yes, please add a check
7	6	Kukla	That fixed it.

Adjacency List strengths

- Easy to inserting a new comment:

```
INSERT INTO Comments (parent_id, author, comment)
VALUES (7, 'Kukla', 'Thanks!');
```

- Easy to move a subtree to a new position:

```
UPDATE Comments SET parent_id = 3
WHERE comment_id = 6;
```


Adjacency List strengths

- Querying a node's children is easy:

```
SELECT * FROM Comments c1  
LEFT JOIN Comments c2  
ON (c2.parent_id = c1.comment_id);
```

- Querying a node's parent is easy:

```
SELECT * FROM Comments c1  
JOIN Comments c2  
ON (c1.parent_id = c2.comment_id);
```


Adjacency List problems

- Hard to query all descendants in a deep tree:

```
SELECT * FROM Comments c1  
LEFT JOIN Comments c2 ON (c2.parent_id = c1.comment_id)  
LEFT JOIN Comments c3 ON (c3.parent_id = c2.comment_id)  
LEFT JOIN Comments c4 ON (c4.parent_id = c3.comment_id)  
LEFT JOIN Comments c5 ON (c5.parent_id = c4.comment_id)  
LEFT JOIN Comments c6 ON (c6.parent_id = c5.comment_id)  
LEFT JOIN Comments c7 ON (c7.parent_id = c6.comment_id)  
LEFT JOIN Comments c8 ON (c8.parent_id = c7.comment_id)  
LEFT JOIN Comments c9 ON (c9.parent_id = c8.comment_id)  
LEFT JOIN Comments c10 ON (c10.parent_id = c9.comment_id)
```

...

*it still doesn't support
unlimited depth!*

Path Enumeration

- Store chain of ancestors in each node 
*good for
breadcrumbs*

comment_id	path	author	comment
1	1/	Fran	What's the cause of this bug?
2	1/2/	Ollie	I think it's a null pointer.
3	1/2/3/	Fran	No, I checked for that.
4	1/4/	Kukla	We need to check valid input.
5	1/4/5/	Ollie	Yes, that's a bug.
6	1/4/6/	Fran	Yes, please add a check
7	1/4/6/7/	Kukla	That fixed it.

Path Enumeration strengths

- Easy to query ancestors of comment #7:

```
SELECT * FROM Comments  
WHERE '1/4/6/7/' LIKE path || '%';
```

- Easy to query descendants of comment #4:

```
SELECT * FROM Comments  
WHERE path LIKE '1/4/%';
```


Path Enumeration strengths

- Easy to add child of comment 7:

```
INSERT INTO Comments (author, comment)
VALUES ('Ollie', 'Good job!');
```

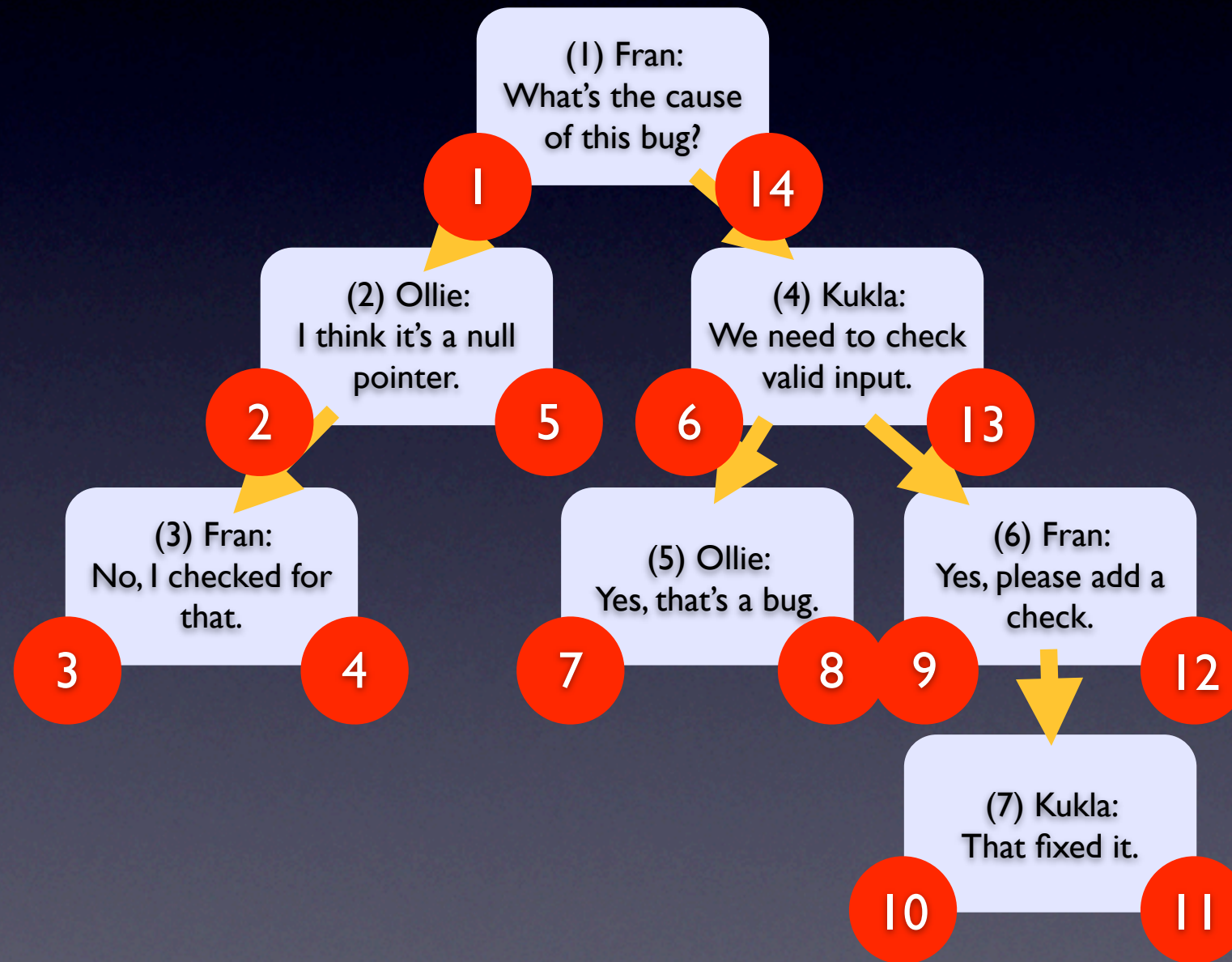
```
SELECT path FROM Comments
WHERE comment_id = 7;
```

```
UPDATE Comments
SET path = $parent_path || LAST_INSERT_ID() || '/'
WHERE comment_id = LAST_INSERT_ID();
```


Nested Sets

- Each comment encodes its descendants using two numbers:
 - A comment's **left** number is **less than** all numbers used by the comment's descendants.
 - A comment's **right** number is **greater than** all numbers used by the comment's descendants.
 - A comment's numbers are **between** all numbers used by the comment's ancestors.

Nested Sets illustration



Nested Sets example

comment_id	nsleft	nsright	author	comment
1	1	14	Fran	What's the cause of this bug?
2	2	5	Ollie	I think it's a null pointer.
3	3	4	Fran	No, I checked for that.
4	6	13	Kukla	We need to check valid input.
5	7	8	Ollie	Yes, that's a bug.
6	9	12	Fran	Yes, please add a check
7	10	11	Kukla	That fixed it.

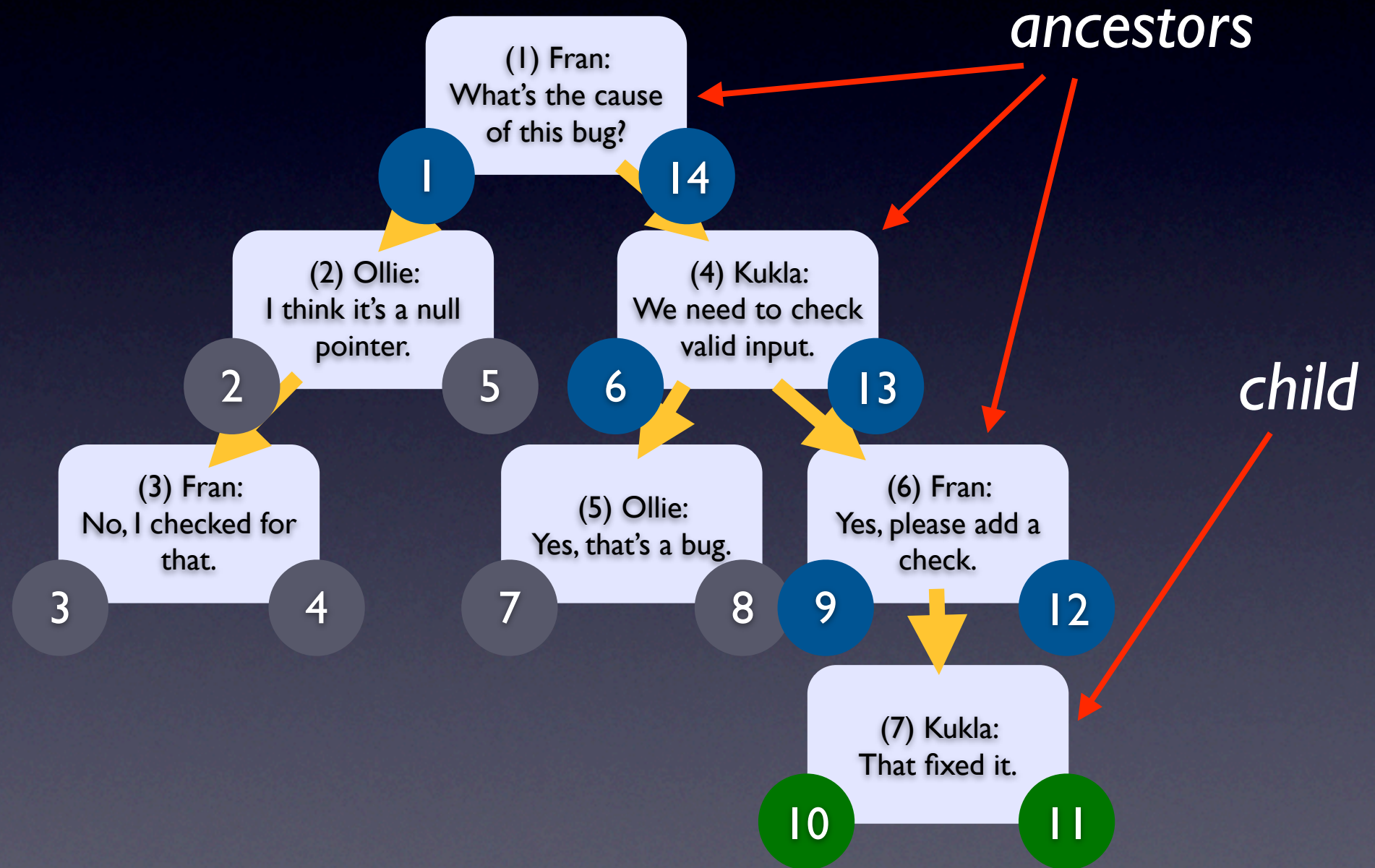
*these are not
foreign keys*

Nested Sets strengths

- Easy to query all ancestors of comment #7:

```
SELECT * FROM Comments child
JOIN Comments ancestor
  ON (child.nsleft BETWEEN ancestor.nsleft
      AND ancestor.nsright)
WHERE child.comment_id = 7;
```


Nested Sets strengths

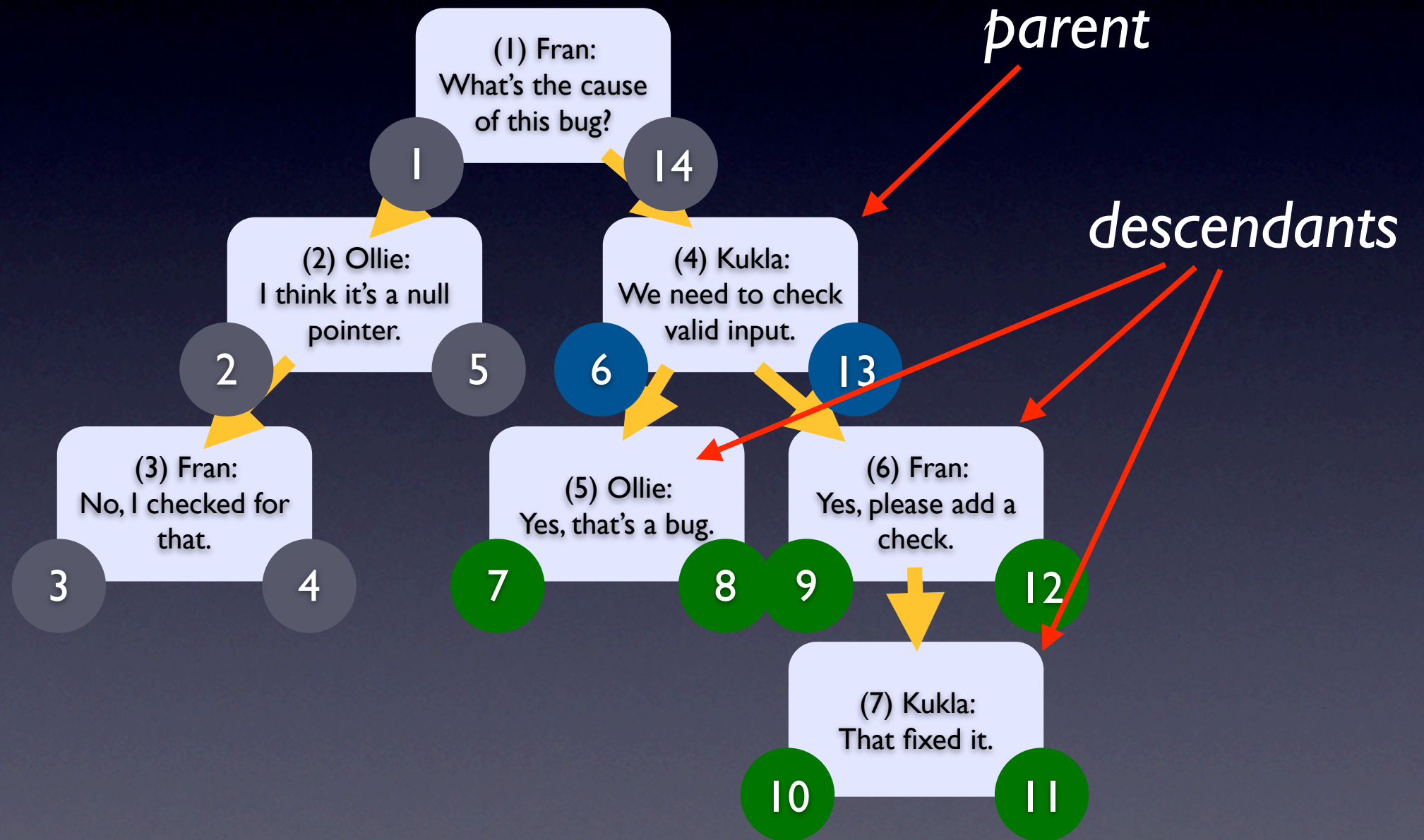


Nested Sets strengths

- Easy to query descendants of comment #4:

```
SELECT * FROM Comments parent
JOIN Comments descendant
  ON (descendant.nsleft BETWEEN parent.nsleft
                                     AND parent.nsright)
WHERE parent.comment_id = 4;
```


Nested Sets strengths



Nested Sets problems

- Hard to insert a new child of comment #5:

```
UPDATE Comments
```

```
SET nsleft = CASE WHEN nsleft >= 8 THEN nsleft+2
```

```
    ELSE nsleft END,
```

```
    nsright = nsright+2
```

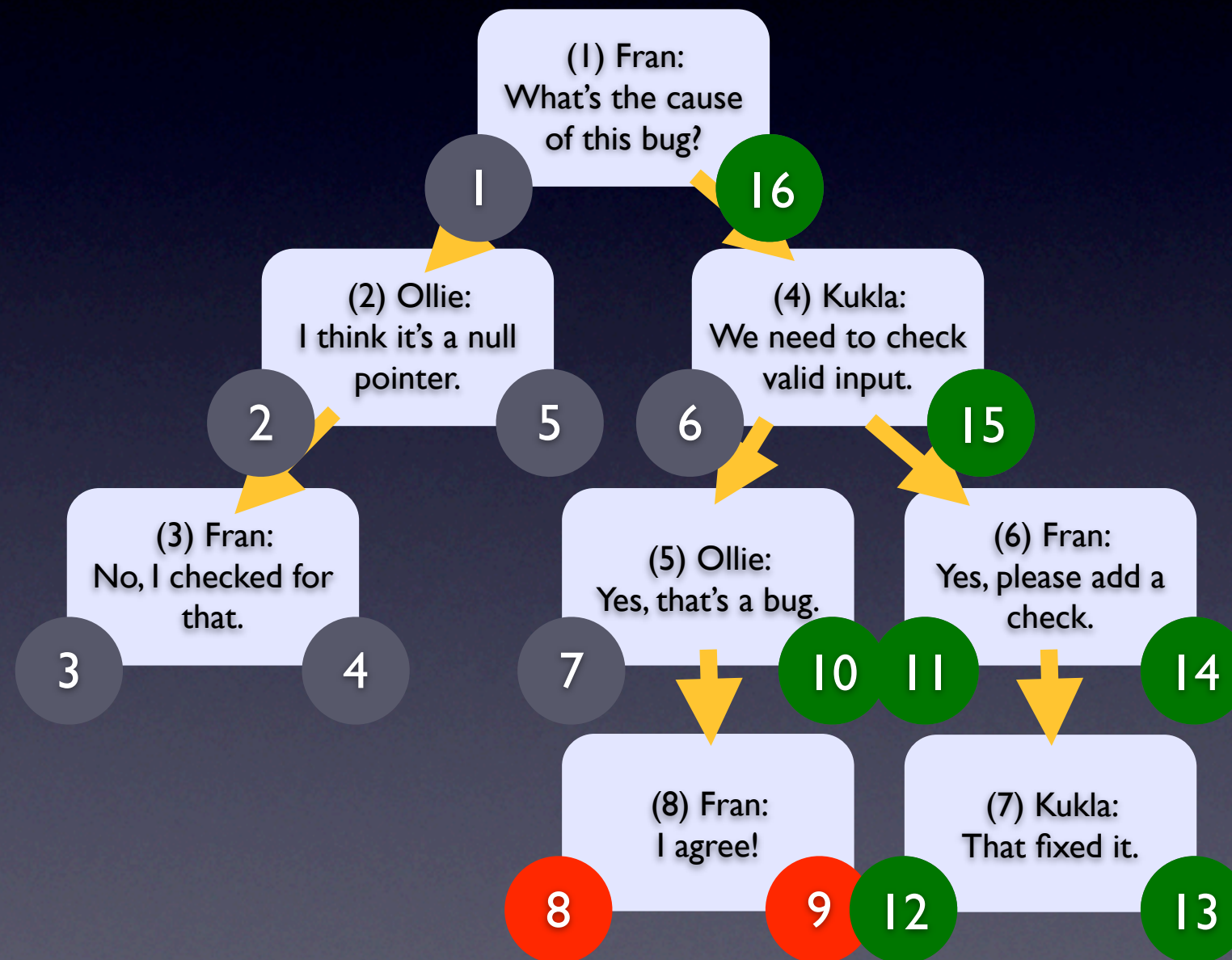
```
WHERE nsright >= 7;
```

```
INSERT INTO Comments (nsleft, nsright, author, comment)
```

```
VALUES (8, 9, 'Fran', 'I agree!');
```

- Recalculate *left* values for all nodes to the right of the new child. Recalculate *right* values for all nodes above and to the right.

Nested Sets problems



Nested Sets problems

- Hard to query the parent of comment #6:

```
SELECT parent.* FROM Comments AS c
JOIN Comments AS parent
  ON (c.nsleft BETWEEN parent.nsleft AND parent.nsright)
LEFT OUTER JOIN Comments AS in_between
  ON (c.nsleft BETWEEN in_between.nsleft AND in_between.nsright
      AND in_between.nsleft BETWEEN parent.nsleft AND parent.nsright)
WHERE c.comment_id = 6 AND in_between.comment_id IS NULL;
```

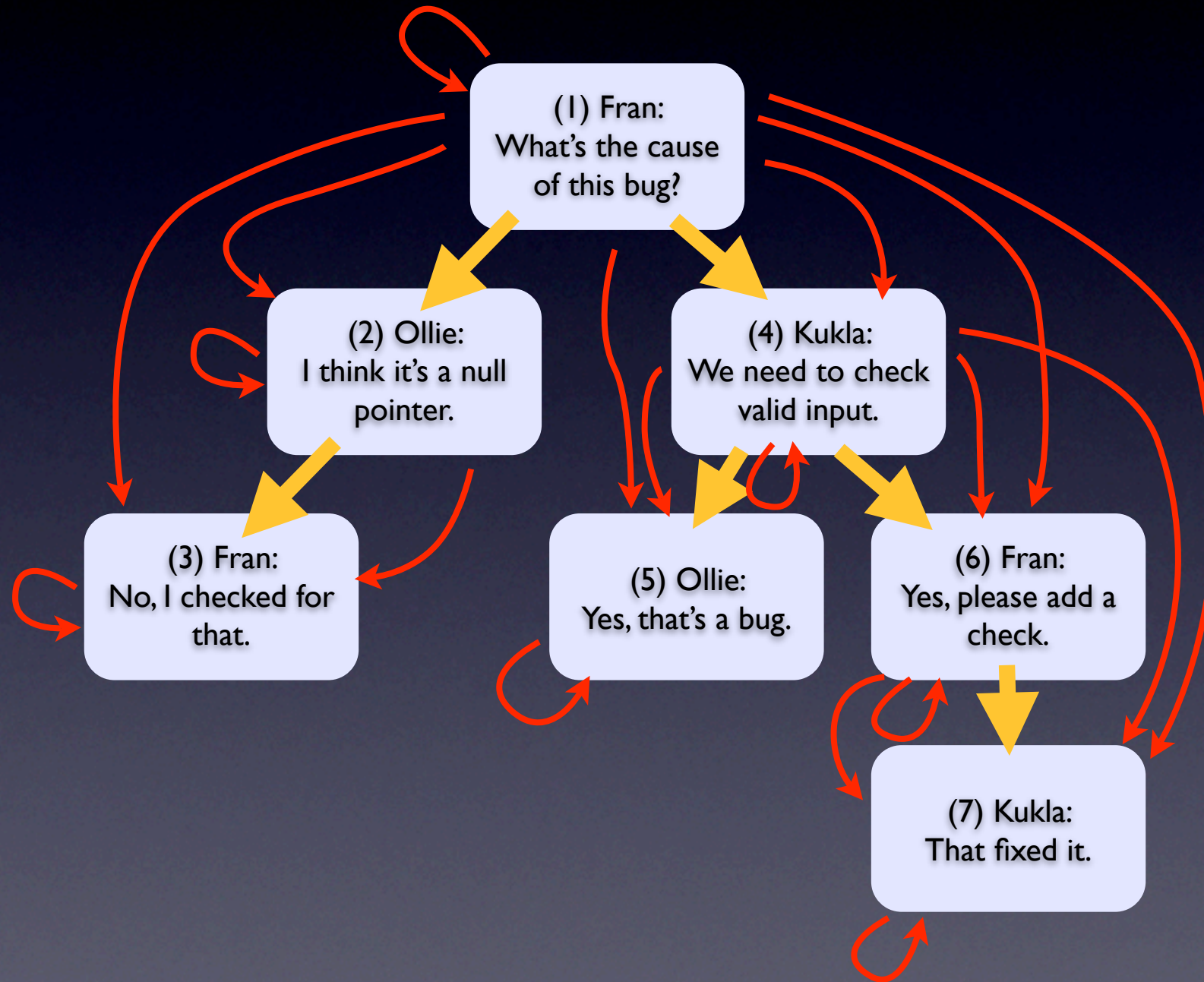
- Parent of #6 is an ancestor who has no descendant who is also an ancestor of #6.
- Querying a child is a similar problem.

Closure Tables

- Store every path from ancestors to descendants
- Requires an additional table:

```
CREATE TABLE TreePaths (  
    ancestor    BIGINT NOT NULL,  
    descendant  BIGINT NOT NULL,  
    PRIMARY KEY (ancestor, descendant),  
    FOREIGN KEY(ancestor)  
        REFERENCES Comments(comment_id),  
    FOREIGN KEY(descendant)  
        REFERENCES Comments(comment_id),  
);
```


Closure Tables illustration



Closure Tables example

comment_id	author	comment
1	Fran	What's the cause of this bug?
2	Ollie	I think it's a null pointer.
3	Fran	No, I checked for that.
4	Kukla	We need to check valid input.
5	Ollie	Yes, that's a bug.
6	Fran	Yes, please add a check
7	Kukla	That fixed it.

requires $O(n^2)$ rows
(but far fewer in practice)

ancestor	descendant
1	1
1	2
1	3
1	4
1	5
1	6
1	7
2	2
2	3
3	3
4	4
4	5
4	6
4	7
5	5
6	6
6	7
7	7

Closure Tables strengths

- Easy to query descendants of comment #4:

```
SELECT c.* FROM Comments c
JOIN TreePaths t
  ON (c.comment_id = t.descendant)
WHERE t.ancestor = 4;
```


Closure Tables strengths

- Easy to query ancestors of comment #6:

```
SELECT c.* FROM Comments c
JOIN TreePaths t
  ON (c.comment_id = t.ancestor)
WHERE t.descendant = 6;
```


Closure Tables strengths

- Easy to insert a new child of comment #5:

```
INSERT INTO Comments  
VALUES (8, 'Fran', 'I agree!');
```

```
INSERT INTO TreePaths (ancestor, descendant)  
SELECT ancestor, 8 FROM TreePaths  
WHERE descendant = 5  
UNION ALL SELECT 8, 8;
```


Closure Tables strengths

- Easy to delete a child comment #7:

DELETE FROM TreePaths
WHERE descendant = 7;



*even easier with
ON DELETE
CASCADE*

Closure Tables strengths

- Easy to delete subtree under comment #4:

```
DELETE FROM TreePaths WHERE descendant IN  
(SELECT descendant FROM TreePaths  
WHERE ancestor = 4);
```

- For MySQL, use multi-table DELETE syntax:

```
DELETE p FROM TreePaths p  
JOIN TreePaths a USING (descendant)  
WHERE a.ancestor = 4;
```


Closure Tables depth

- Add a *depth* column to make it easier to query immediate parent or child:

```
SELECT c.* FROM Comments c
JOIN TreePaths t
  ON (c.comment_id = t.descendant)
WHERE t.ancestor = 4
      AND t.depth = 1;
```

ancestor	descendant	depth
1	1	0
1	2	1
1	3	2
1	4	1
1	5	2
1	6	2
1	7	3
2	2	0
2	3	1
3	3	0
4	4	0
4	5	1
4	6	1
4	7	2
5	5	0
6	6	0
6	7	1
7	7	0

Choose the right design

Design	Tables	Query Child	Query Subtree	Delete Node	Add Node	Move Subtree	Referential Integrity
Adjacency List	1	Easy	Hard	Easy	Easy	Easy	Yes
Path Enumeration	1	Easy	Easy	Easy	Easy	Easy	No
Nested Sets	1	Hard	Easy	Hard	Hard	Hard	No
Closure Table	2	Easy	Easy	Easy	Easy	Hard	Yes

Magic Beans

Essentially, all models are wrong, but some are useful.

— George E. P. Box

Magic Beans

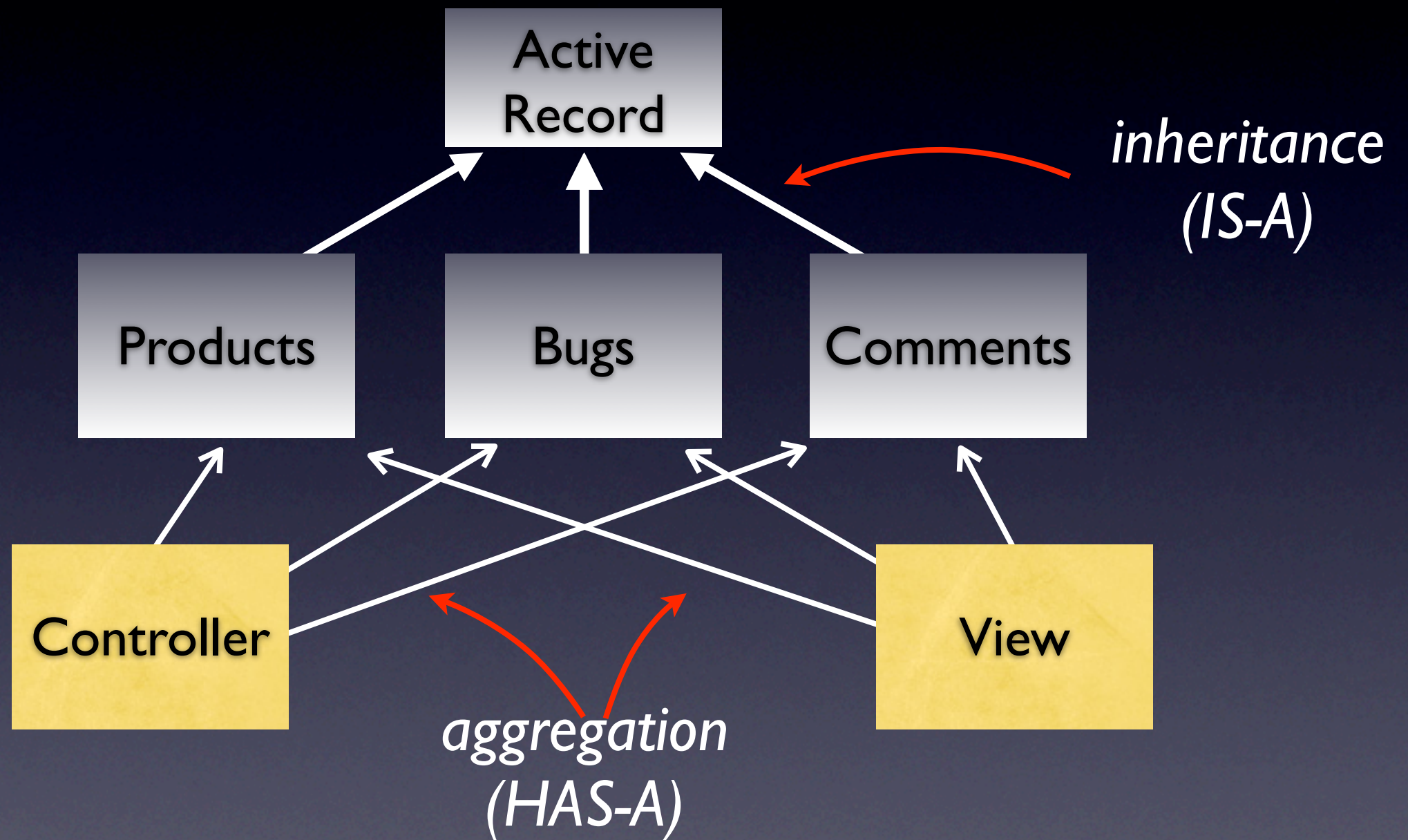
- **Objective:**
 - Model-View-Controller application development
 - Object-Relational Mapping (ORM)

Model = Active Record?



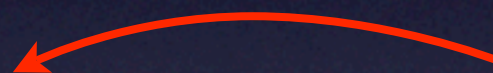

- The *Golden Hammer* of data access
- “Model” misused in MVC frameworks:





Model *is-a* Active Record?



Problem: OO design

- “Model” : Active Record  *inheritance
(IS-A)*
- Model tied to database schema  *coupling*
- Product → Bug, or
Bug → Product?  *unclear assignment
of responsibilities*
- Models expose Active Record interface,
not domain-specific interface  *poor
encapsulation*

Problem: MVC design

- Controllers need to know database structure  *“T.M.I.” !!*
- Database changes require code changes  *not “DRY”*
- <http://www.martinfowler.com/bliki/AnemicDomainModel.html>

Problem: testability

- Harder to test Model without database
- Need database “fixtures”
- Fat Controller makes it harder to test business logic

*tests are
slow*

*tests are
even slower*

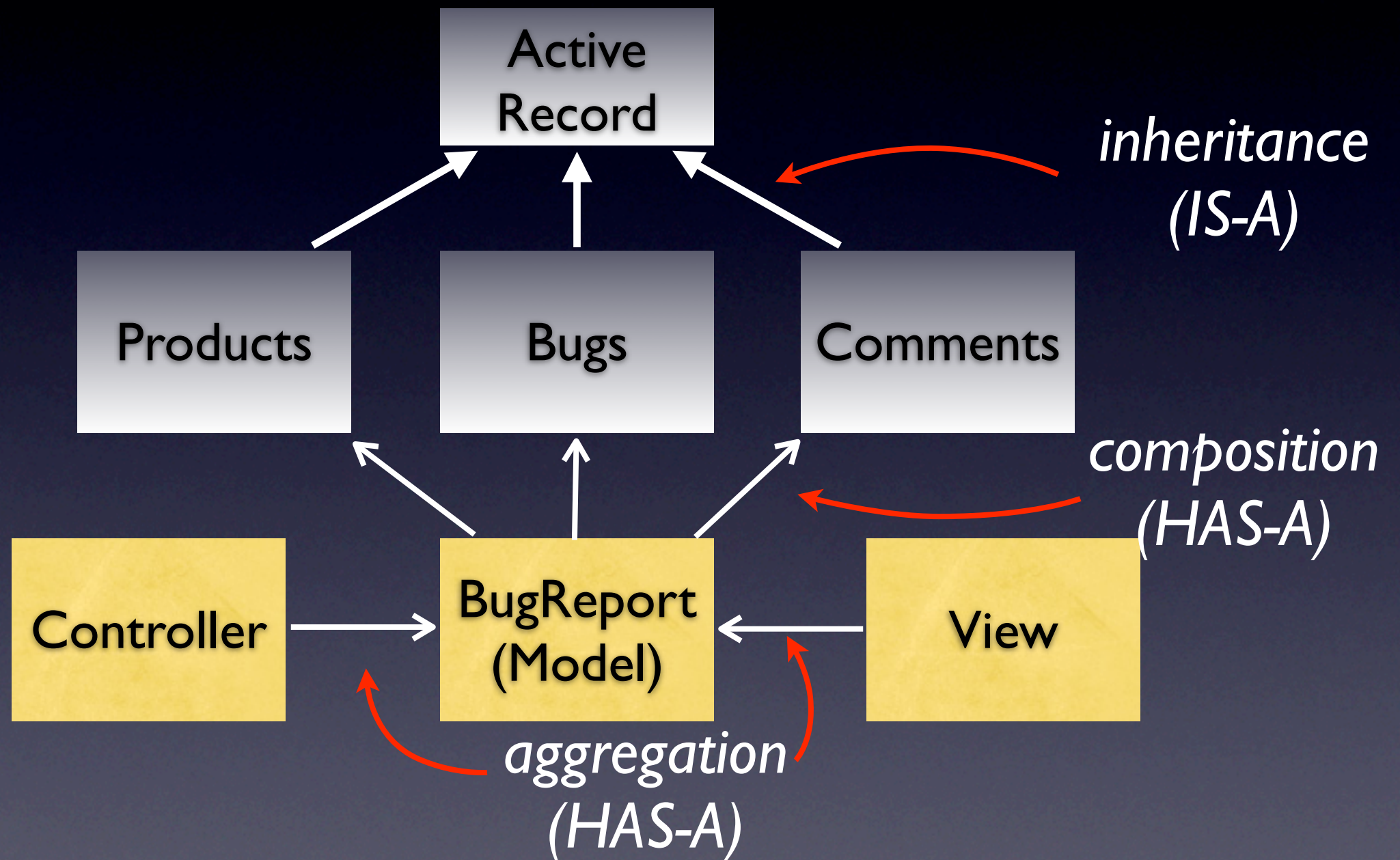
*mocking HTTP Request,
scraping HTML output*

Magic Beans

Solution?

OO is about *decoupling*

Model *has-a* Active Record(s)



Model characteristics

- Extend no base class
- Abstract the database
- Expose only domain-specific interface
- Encapsulate complex business logic

Model benefits

- Models are decoupled from DB
 - Supports mock objects
 - Supports dependency injection
- Unit-testing Models in isolation is faster
- Unit-testing Thin Controllers is easier

Decouple your Models

- You can use this architecture



...even in MVC frameworks that encourage the antipattern.



karwin software solutions

Copyright 2008-2009 Bill Karwin

www.slideshare.net/billkarwin

Released under a Creative Commons 3.0 License:
<http://creativecommons.org/licenses/by-nc-nd/3.0/>

You are free to share - to copy, distribute and
transmit this work, under the following conditions:

Attribution.

You must attribute this
work to Bill Karwin.

Noncommercial.

You may not use this work
for commercial purposes.

No Derivative Works.

You may not alter,
transform, or build
upon this work.

