
Google™
11
I/O

Large-scale Data Analysis Using the App Engine Pipeline API

Brett Slatkin
May 11th, 2011

Hashtags: #io2011 #AppEngine
Feedback: http://goo.gl/1gGbS

Projects:

appengine-pipeline.googlecode.com

mapreduce.appspot.com

Slides:

`http://goo.gl/P2U2S`

Motivating example

Data joins

- What?
  - An SQL-like join, but bigger
  - Combine data by its features
- Why?
  - Generate views, reports, data
  - Examples: Ranking, heat-maps, roll-ups
- When?
  - Storage of each dataset optimized for something else
    - Write latency (logs)
    - Per-user views
  - Lots of data!

Example: Sales report

Datastore kinds

- Sale
  - What was sold to customers
  - Customers, Items (many to many)
- Item
  - How much the item costs
  - Item, cost (one to one)
- Category
  - What kind of thing is the item
  - Items, Categories (many to many)

Example: Sales report

Data

- Customers: A, B, C
- Sales
  - Milkshake: A, B
  - Pie: A, C
  - Fries: B, C
- Categories
  - Desserts: Pie, Milkshake
  - Food: Pie, Fries
  - Drinks: Milkshake
- Items
  - Milkshake: $3
  - Pie: $5
  - Fries: $2

Example: Sales report

Report

- Customer
  - A = Milkshake + Pie = $8
  - B = Milkshake + Fries = $5
  - C = Pie + Fries = $7
- Categories:
  - Desserts = 2*Milkshake + 2*Pie = $16
  - Food = 2*Pie + 2*Fries =$14
  - Drinks = 2*Milkshake = $6

Example: Sales report

```
-- By customer
SELECT Sales.customerId, sum(Items.price) as spend
FROM Sales, Items
WHERE Sales.itemId = Items.itemId
GROUP BY Sales.customerId;

-- By category
SELECT Category.name, sum(Items.price) as spend
FROM Category
INNER JOIN Sales
  ON Sales.itemId = Category.itemId
INNER JOIN Items
  ON (Sales.itemId = Items.itemId)
GROUP BY Category.name;
```

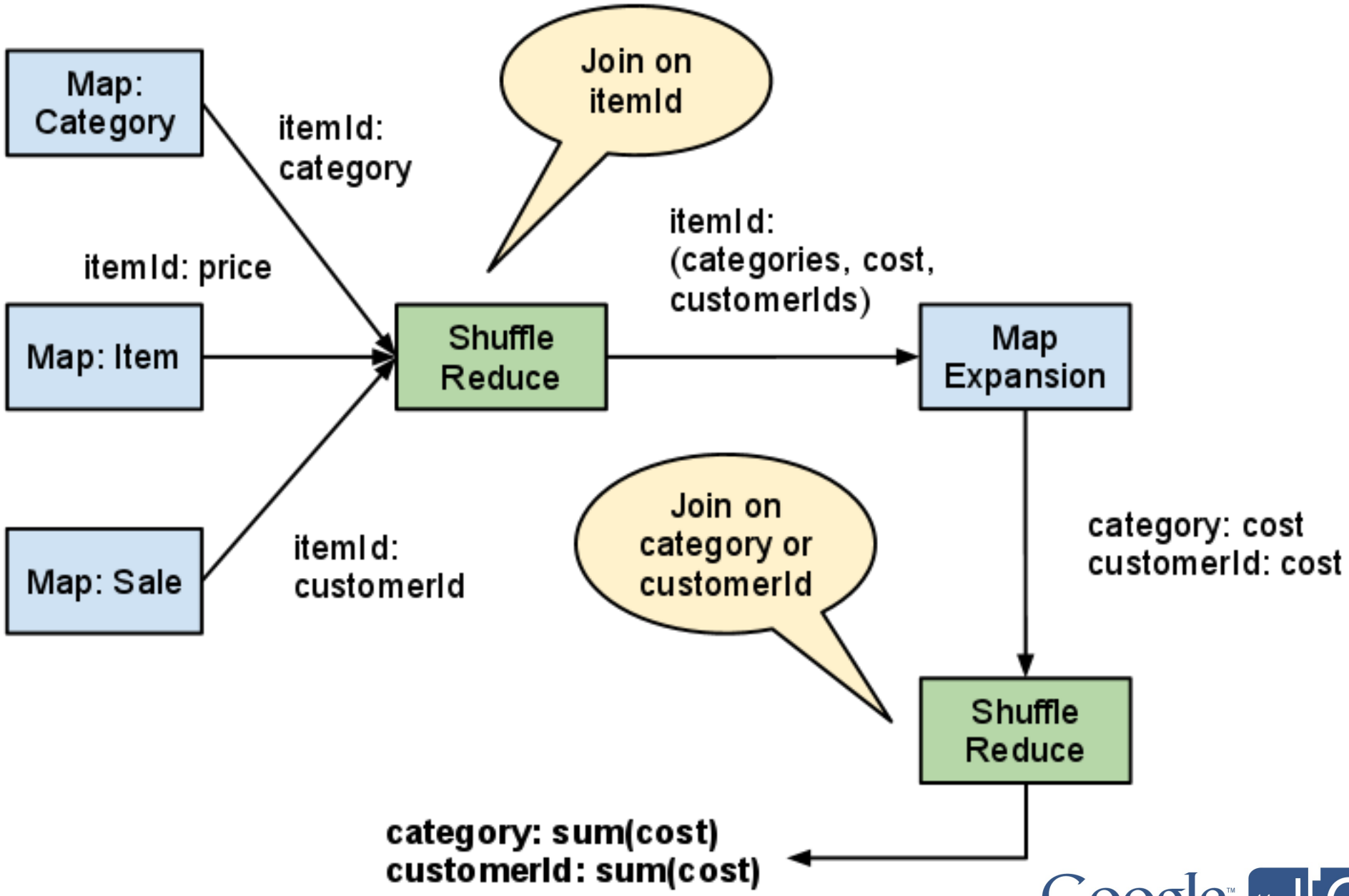
This will break, eventually!

Solution

Map Reduce

- Keeps working at extreme sizes

Example: Sales report



Solution... mostly

Map Reduce

- Keeps working at extreme sizes

But-- Now you have new problems

- How do I connect all of these Map Reduces together?

Turns out--

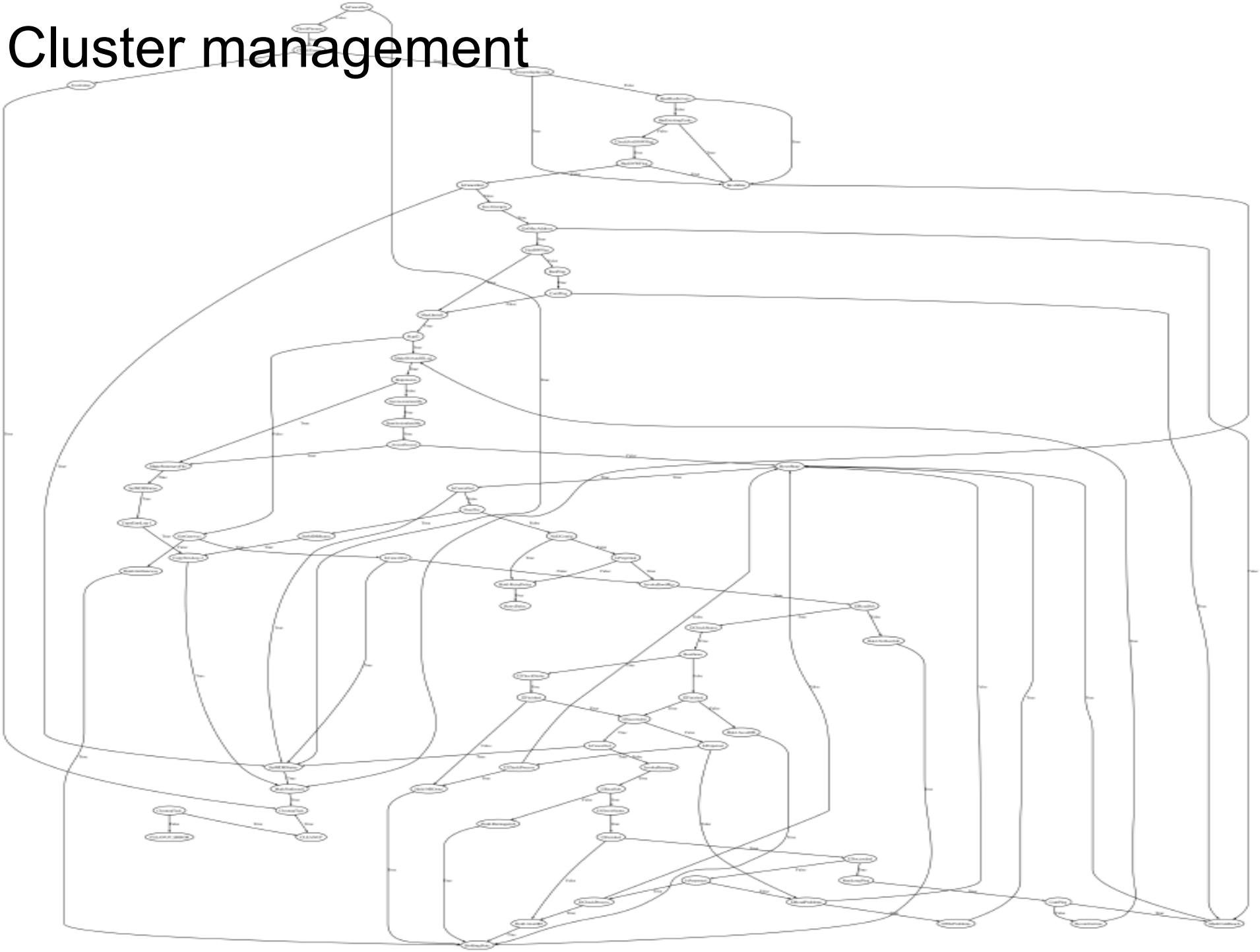
- Fan-in is hard
- Fan-in is a type of workflow

Prior art

Finite state machines

- My experience
  - Cluster management tool
  - App Engine billing system
  - High-performance flows
- Problems
  - Explosion of states
  - Co-development, reuse
  - Testing
- Examples
  - Many out there
  - Fantasm for App Engine

Cluster management



Work queues

- Producer/consumer style queue
- Equal, small chunks
- Well-defined input/output per chunk
- Measurable processing rate

- Problems
  - No control flow
  - Hard to test

- Examples
  - Gearman
  - App Engine queues

Data dependencies

- "make" for multi-stage mapreduces
- Define input, outputs, and rules
- Incremental builds

- Problems
  - No control flow
  - No outside inputs (humans)
  - Black-box testing only

- Examples
  - Many within Google

Flow oriented

- Convert your data into tuples
- Define transforms on tuples
- Automatic optimization
- Efficient for > 1TB of data

- Problems
  - Complex
  - Buy into the religion
  - No glue

- Examples
  - Cascading

How the Pipeline API is different

Assumptions

- Vast number of concurrent threads
- Highly available storage (HR Datastore, Blobstore)
- Elastic scaling (task queue + dynamic instances)
- No config for more resources

Goals

- Non-deterministic
- Functional testing
- Reusable
- Productionized
- Minimal

Non-goals

- Static analysis
- Automatic optimization
- Sustained flow rates
- BPEL translation, etc

Types of work

- Synchronous
  - Up to 10 minutes (task queue limit)
  - As long as you want! (backends)

- Web service calls
- Transactions
- Datastore iterations

Types of work

- Asynchronous
  - Fire off work now, wait for a callback
  - Tie-in with other frameworks (eg, Mapper)

- Map Reduce (hours)
- Send email (eg, click to confirm)
- Webhooks

It's all about Coordination

- Pass around arguments and outputs
- Spawn many child tasks, wait for them
- Make "joining" results of parallel work simple

Making Python and Java Parallel

Slides:

http://goo.gl/P2U2S

Definition style -- Python

```
class Add(pipeline.Pipeline):
  def run(self, a, b):
    return a + b
class Multiply(pipeline.Pipeline):
  def run(self, a, b):
    return a * b

class LinearFunc(pipeline.Pipeline):
  def run(self, x, slope=1, offset=0):
    # y = m*x + b
    mx = yield Multiply(x, slope)
    yield Add(mx, offset)
```

Invocation style -- Python

```
# Create it like an object
job = LinearFunc(6, slope=3.5, offset=7)
job.start()
pipeline_id = job.pipeline_id

# Access outputs later
job = LinearFunc.from_id(pipeline_id)
if job.has_finalized:
  job.outputs.default.value == 28  # True
```

Testing style -- Python

```
class MyPipeline(pipeline.Pipeline):
  def run(self, x):
    yield TakesForeverChild(x)
  def run_test(self, x):
    return x * 3  # Synchronous mock/fake

# To test
job = MyPipeline(723)
job.start_test()  # Runs serially!
job.outputs.default.value == 2169  # True
```

Definition style -- Java

```
class Add extends Job2<Double, Double, Double> {
  public Value<Double> run(Double a, Double b) {
    return immediate(a + b);
  }
}
class Multiply extends Job2<Double, Double, Double> {
  public Value<Double> run(Double a, Double b) {
    return immediate(a * b);
  }
}
class LinearFunc extends
    Job3<Double, Double, Double, Double> {
  public Value<Double> run(
      Double x, Double m, Double b) {
    FutureValue<Double> mx = futureCall(
      new Multiply(), immediate(m), immediate(x));
    return futureCall(new Add(), mx, immediate(b));
  }
}
```

Invocation style -- Java

```
//Run the pipeline
PipelineService service =
  PipelineServiceFactory.newPipelineService();
String pipelineId = service.startNewPipeline(
  new LinearFunc(), 3.5, 6.0, 7.0);
// Access outputs later
JobInfo jobInfo = service.getJobInfo(pipelineId);
if (jobInfo.getjobState() ==
    JobInfo.State.COMPLETED_SUCCESSFULLY) {
  System.out.println("Result: " + jobInfo.getOutput());
}
```

Future values -- Python

```
class Multiply(pipeline.Pipeline):
  def run(self, a, b):
    return a * b

class MyPipeline(pipeline.Pipeline):
  def run(self, x, y):
    result = yield Multiply(x, y)
    logging.info('Value: %s', result)   # No
    r.outputs.default.value             # No
    r.wait()                            # No
```

Future values -- Python

```
class Multiply(pipeline.Pipeline):
  def run(self, a, b):
    return a * b
class LogValue(pipeline.Pipeline):
  def run(self, value):
    logging.info('Value: %s', value)

class MyPipeline(pipeline.Pipeline):
  def run(self, x, y):
    result = yield Multiply(x, y)
    yield LogValue(result)  # Works
```

Immediate values -- Python

```
class CheckThreshold(pipeline.Pipeline):
  def run(self, value, threshold, message):
    if value <= threshold:                # Yes
      return
    logging.info(
      '%s > %s: %s',
      value, threshold, message)  # Yes
```

Future values -- Java

```
class Multiply extends Job2<Double, Double, Double> {
  public Value<Double> run(Double a, Double b) {
    return immediate(a * b);
  }
}
class MyPipeline extendsJob2<Double, Double Double> {
  public Value<Double> run(Double x, Double y) {
    FutureValue<Double> result = futureCall(
      new Multiply(), immediate(x), immediate(y));
    System.out.println(
      "Value: " + result.getValue()); // No
  }
}
```

Future values -- Java

```
class Multiply extends Job2<Double, Double, Double> {
  public Value<Double> run(Double a, Double b) {
    return immediate(a * b);
  }
}
class PrintIt extends Job1<Void, Double> {
  public Value<Void> run(Double value) {
    System.out.println("Value: " + value);
  }
}
class MyPipeline extendsJob2<Double, Double Double> {
  public Value<Double> run(Double x, Double y) {
    FutureValue<Double> result = futureCall(
      new Multiply(), immediate(x), immediate(y));
    futureCall(new PrintIt(), result);  // Works
  }
}
```

Immediate values -- Java

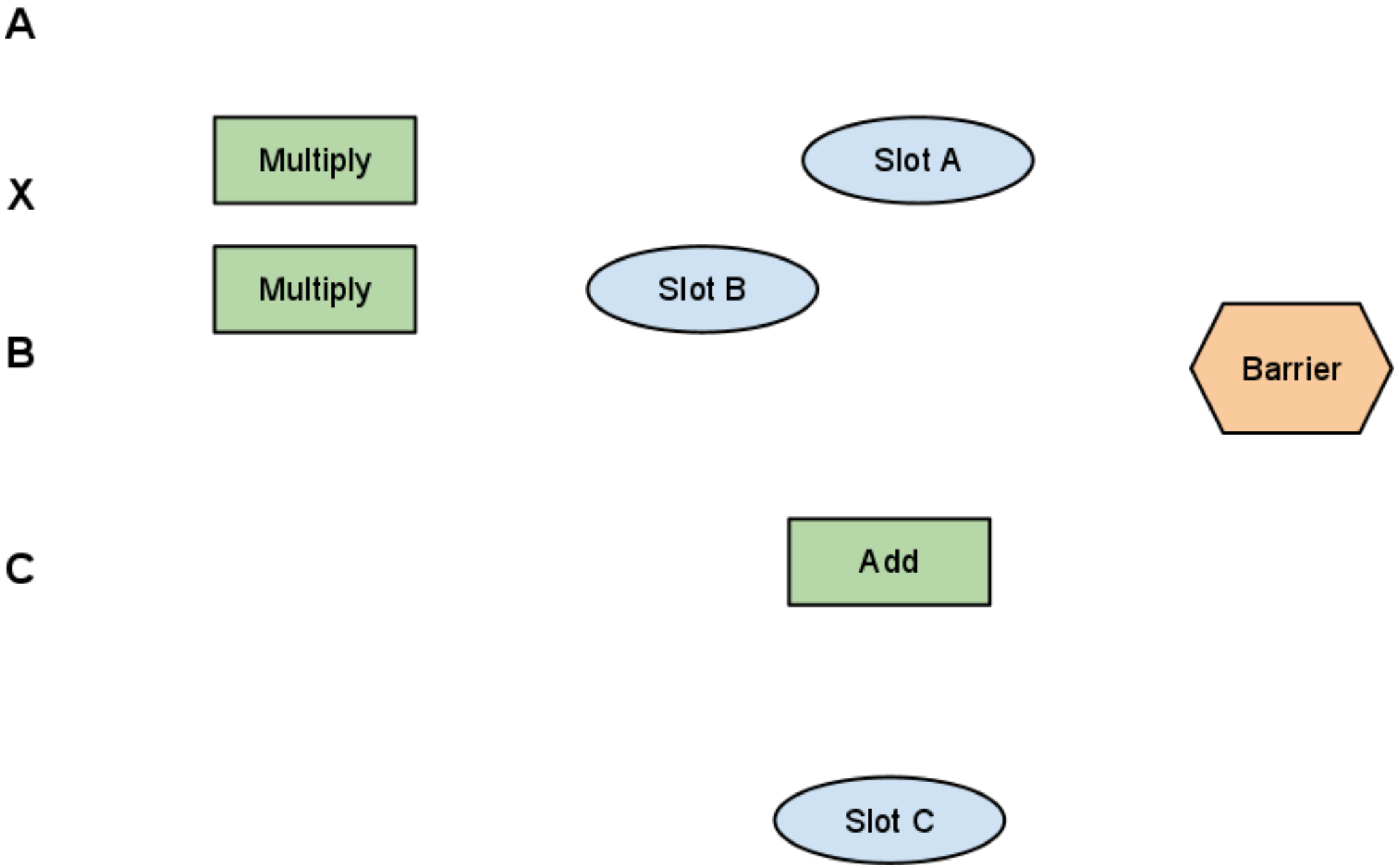
```
class Check extends Job3<Void, Double, Double, String> {
  public Value<Void> run(Double value, Double threshold,
                         String message) {
    if (value > threshold) {
      System.out.println(message + ": " + value +
                         " > " + threshold);  // Works
    }
  }
}
class MyPipeline extendsJob2<Void, Double Double> {
  public Value<Void> run(Double x, Double y) {
    futureCall(
      new Check(), immediate(x), immediate(y));  // Works
  }
}
```

Why it works: Call graph tracing

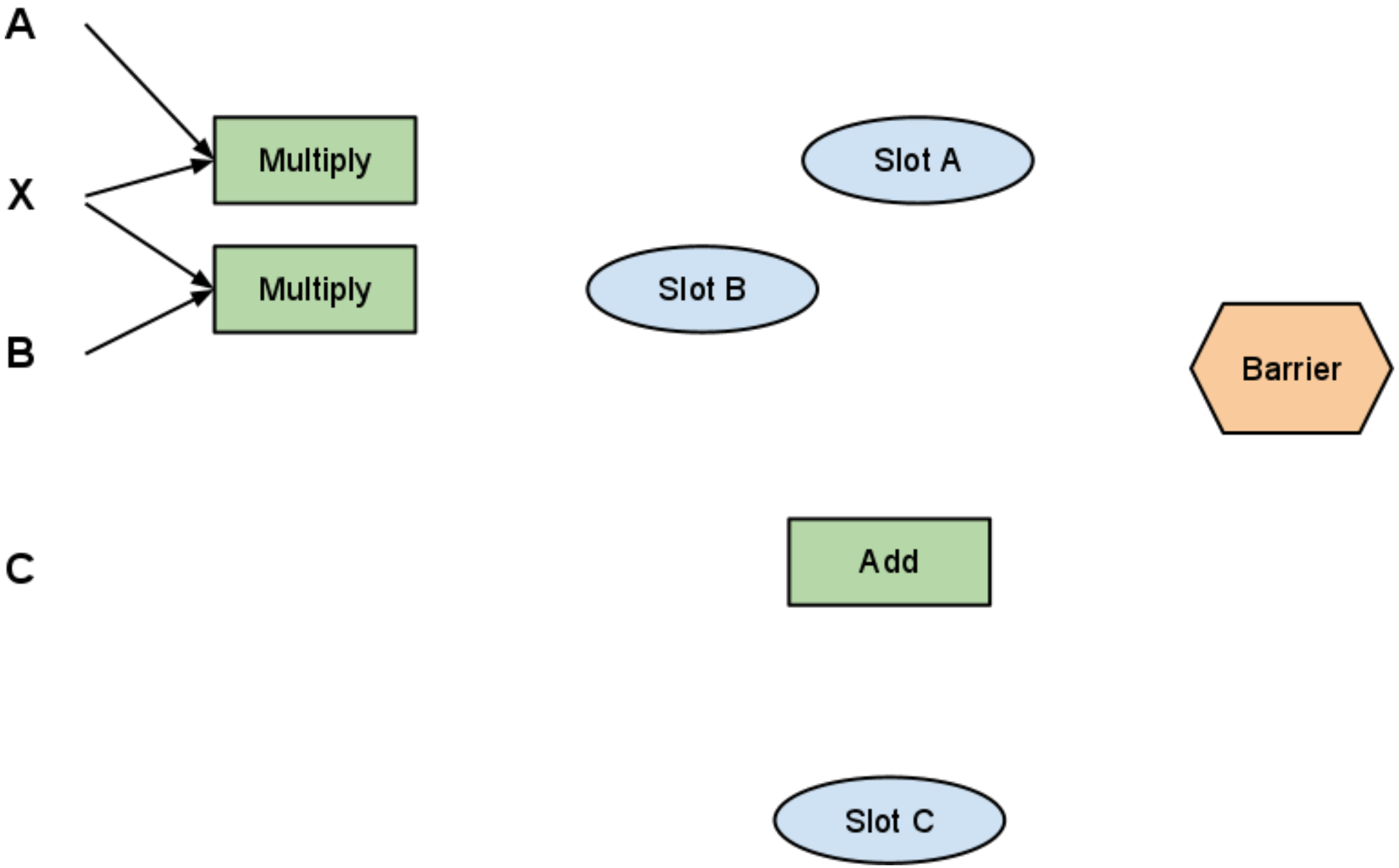
Why it works: Call graph tracing

```
class Add(pipeline.Pipeline):
  def run(self, *values):
    return sum(values)

class Multiply(pipeline.Pipeline):
  def run(self, i=1, j=1, k=1):
    return i * j * k

class Polynomial(pipeline.Pipeline):
  def run(self, x, A, B, C):
    # y = A*x^2 + B*x + C
    Ax_2 = yield Multiply(A, x, x)
    Bx = yield Multiply(B, x)
    yield Add(Ax_2, Bx, C)
```

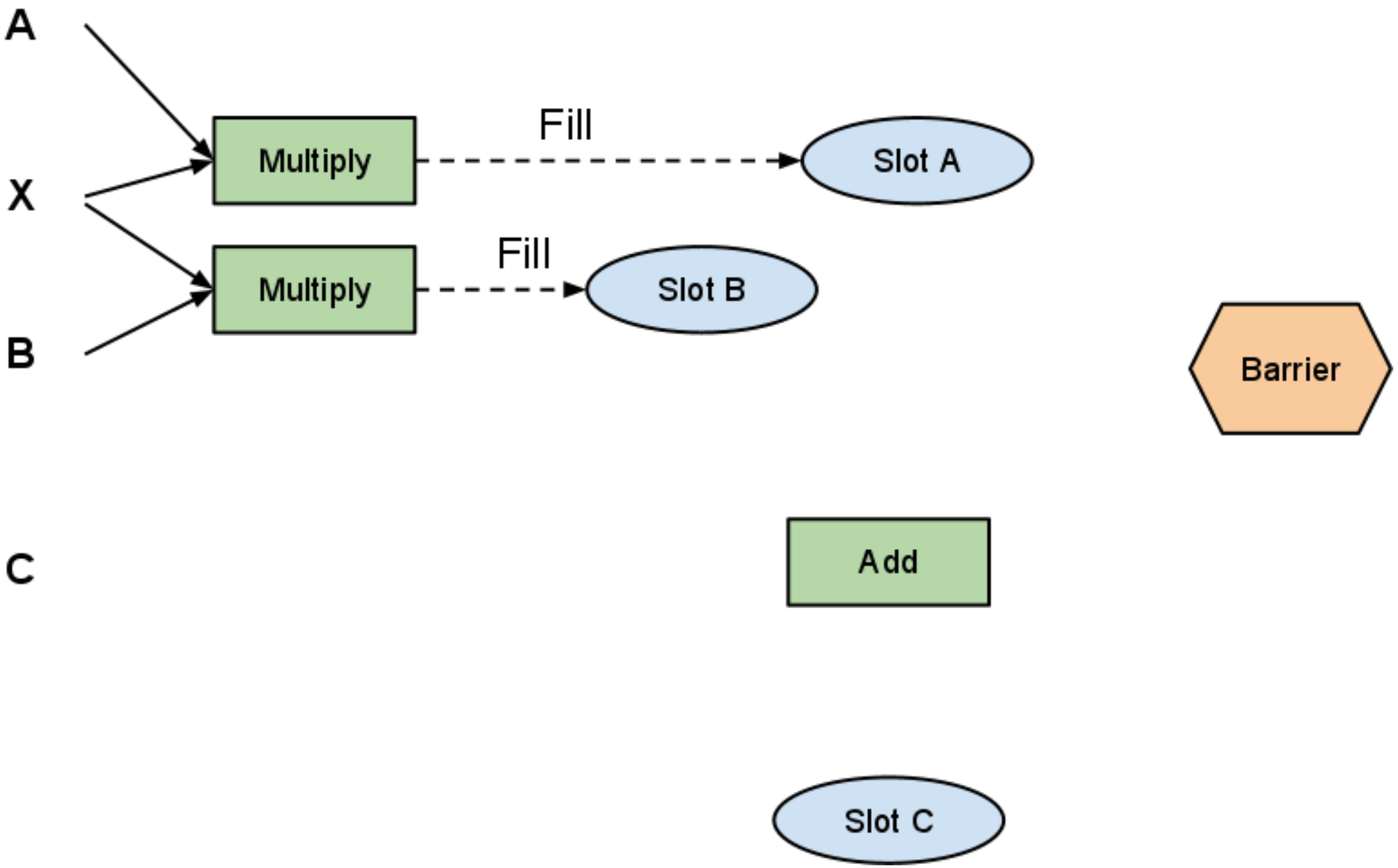
Why it works: Call graph tracing



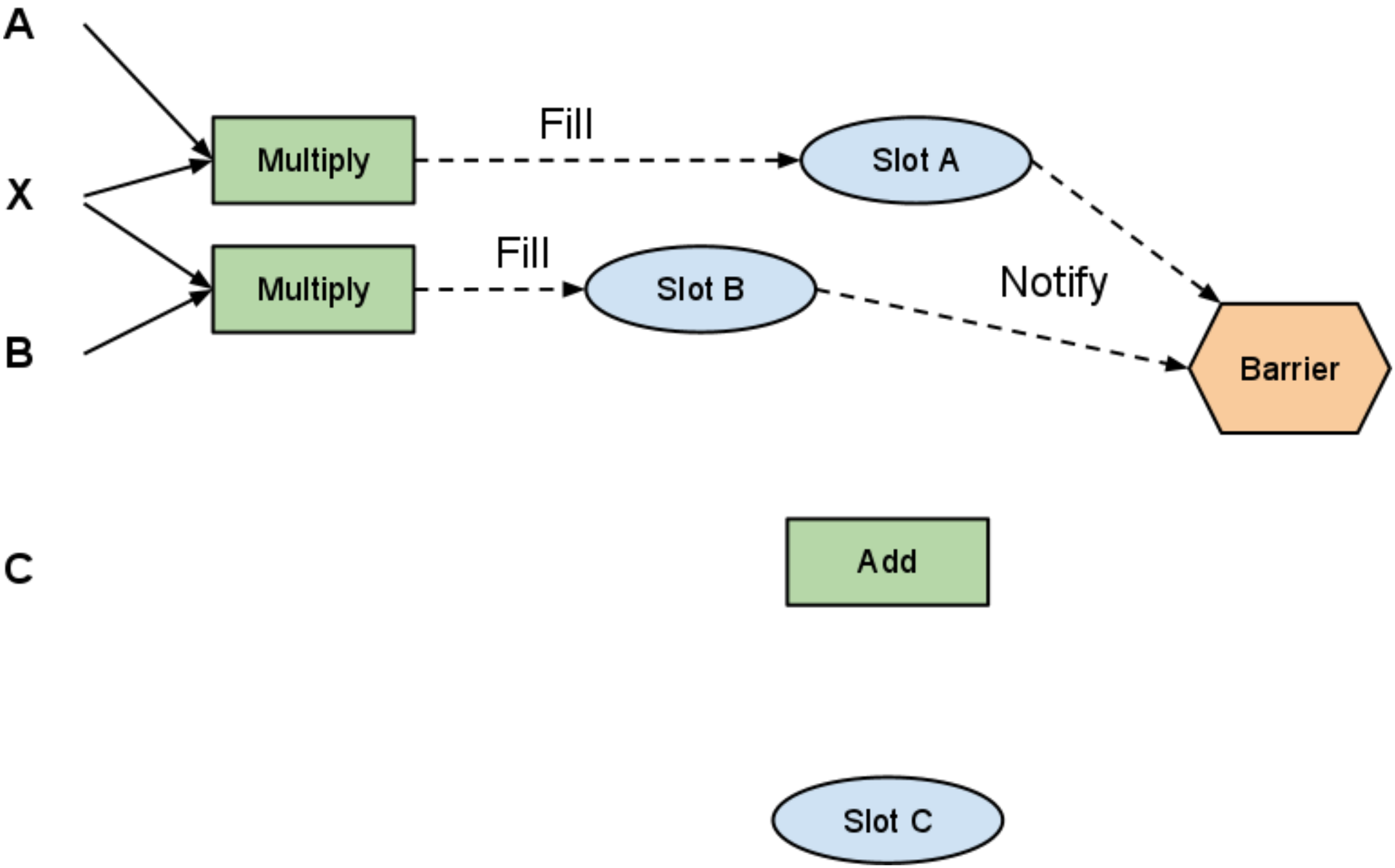
Why it works: Call graph tracing



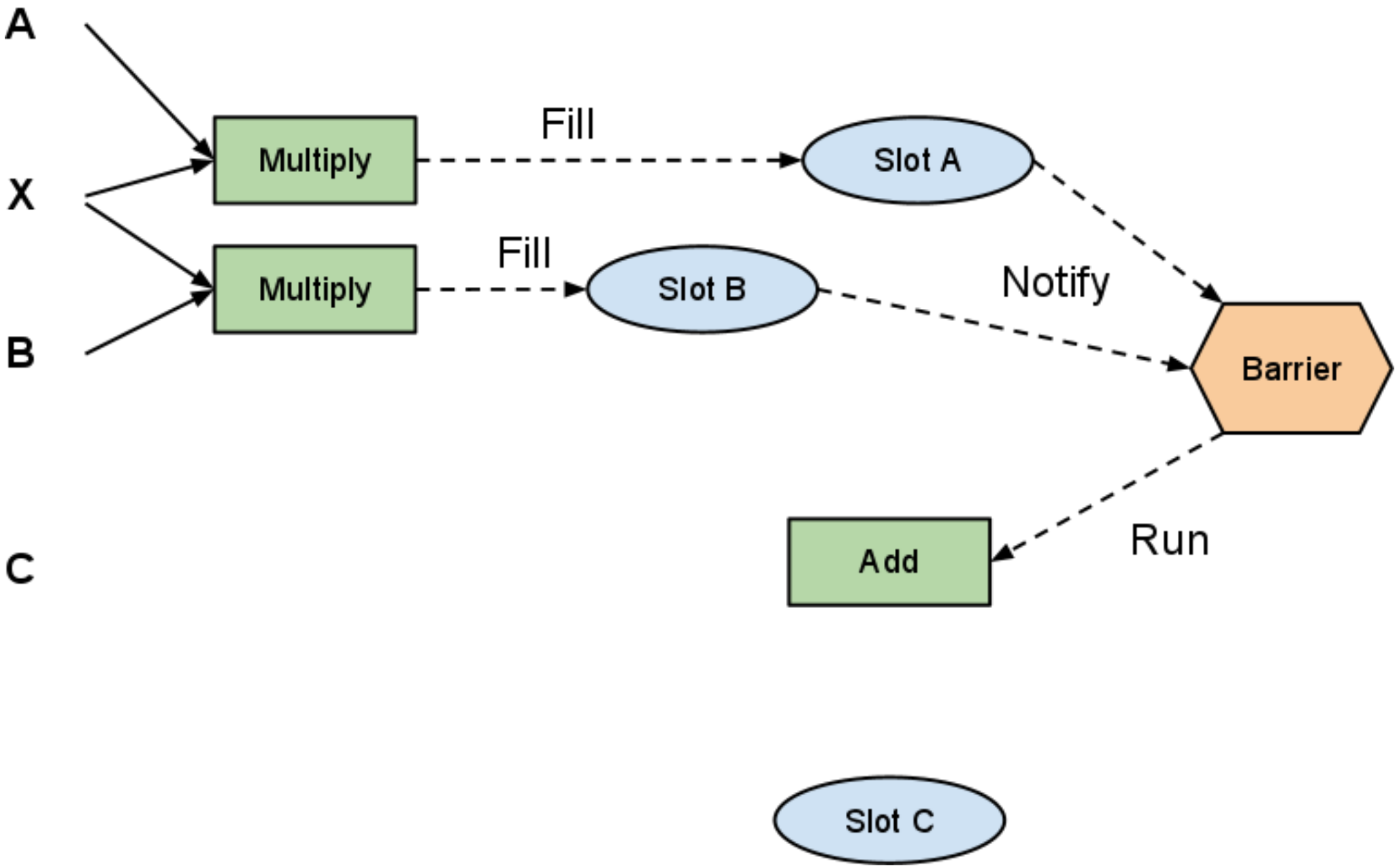
Why it works: Call graph tracing



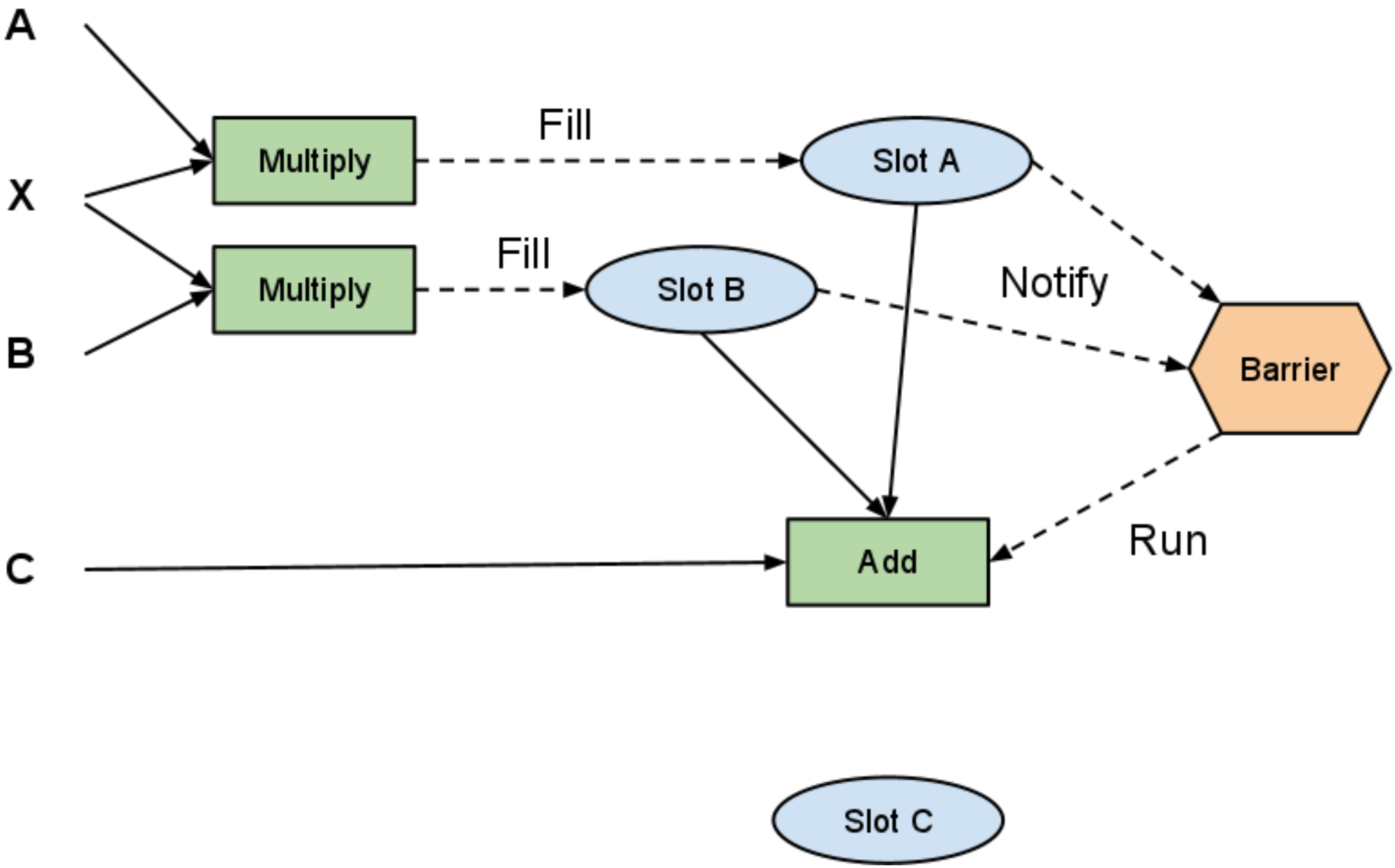
Why it works: Call graph tracing



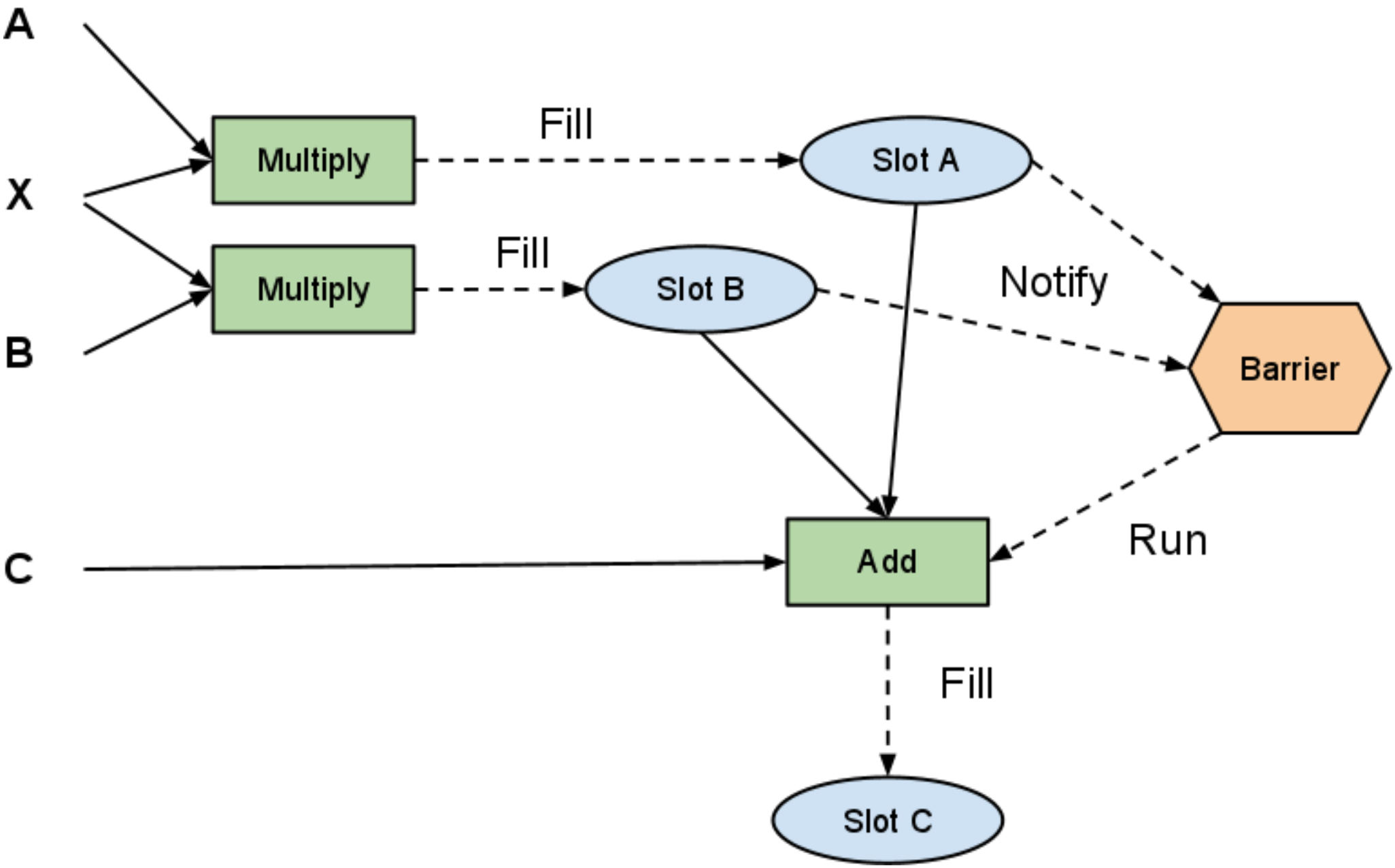
Why it works: Call graph tracing



Why it works: Call graph tracing



Why it works: Call graph tracing



Demo

Fan-out -- Python

```
class SendEmail(pipeline.Pipeline):
  def run(self, from, to, body):
    mail.send(from, to, body)
class EmailInvites(pipeline.Pipeline):
  def run(self, event_id):
    e = Event.get_by_id(event_id)
    for to in e.invited:
      yield SendEmail(e.from, to, e.body)
    # All children run immediately
```

Fan-in, coordination -- Python

```
class WordCountUrl(pipeline.Pipeline):
  def run(self, url):
    r = urlfetch.fetch(url)
    return len(r.data.split())

class Sum(pipeline.Pipeline):
  def run(self, *values):
    return sum(values)

class MyPipeline(pipeline.Pipeline):
  def run(self, *urls):
    results = []
    for u in urls:
      r.append((yield WordCountUrl(u))
    yield Sum(*results) # Barrier waits
```

Sequencing -- Python

```
class MyPipeline(pipeline.Pipeline):
  def run(self, a, b):
    a = JobA(...)
    b = JobB(...)

    with pipeline.After(a, b):
      yield CleanupFiles(a, b)

    with pipeline.InOrder():
      yield UpdateDashboard()
      yield EmailTeam()
```

Fan-out -- Java

```
class SendEmail extends
    Job3<Void, String, String, String> {
  public Value<Void> run(String from, String to,
                         String body) {
    getMailService().send(
      new Message(from, to, "Invite", body));
  }
}
class SendInvites extends Job2<Void, Invite> {
  public Value<Void> run(Invite invite) {
    for (String to : invite.getInvitedList()) {
      futureCall(new SendEmail(),
        immediate(invite.getFrom()),
        immediate(to),
        immediate(invite.getBody()));
    }
  }
}
```

Fan-in, coordination -- Java

```
class WordCountUrl extends Job1<Integer, String> {
  public Value<Void> run(String url) {
    return new String(getURLFetchService()
      .fetch(url).getContent(), "UTF-8").split(" ");
  }
}

class Sum extends Job1<Integer, List<Integer>> {
  public Value<Integer> run(List<Integer> values) {
    int total = 0;
    for (int v : values) { total+= v; }
    return immediate(total);
  }
}
```

Fan-in, coordination 2 -- Java

```
class MyPipeline extends Job2<Integer, List<String>> {
  public Value<Integer> run(List<String> urls) {
    FutureValue<Integer>[] results =
      new FutureValue<Integer>[urls.size()];

    for (int i = 0; i < urls.size(); ++i) {
      results[i] = futureCall(
        new WordCountUrl(), immediate(urls.get(i)));
    }
    return futureCall(new Sum(), futureList(results));
  }
}
```

Sequencing -- Java

```
class ThisThenThat extends
    Job3<Void, String, String> {
  public Value<Void> run(String a, String b) {
    FutureValue<Void> jobA = futureCall(
      new JobA(), immediate(a));
    futureCall(new JobB(), waitFor(a), immediate(b));
  }
}
```

Asynchronous -- Python

```
class Delay(pipeline.Pipeline):
  async = True
  def run(self, seconds=None):
    task = self.get_callback_task(
      countdown=seconds,
      name='delay-' + self.cascade_id)
    try:
      task.add(self.queue_name)
    except (TombstonedTaskError,
            TaskAlreadyExistsError):
      pass

  def callback(self):
    self.complete()
```

Asynchronous -- Java

```
class FavoriteColor extends Job1<Void, String> {
  public Value<Void> run(String address) {
    PromisedValue<String> color =
      newPromise(String.class);

    getMailService().send(new Message(
      "sender@example.com", address, "My subject",
      "Here's your token: " + color.getHandle());

    futureCall(new SaveColor(), color);
  }
}

// Some separate agent
submitPromisedValue(colorHandle, theValue);
```

Finalized -- Python

```
class MapReduce(pipeline.Pipeline):
  def run(self, ...):
    m = yield Map(...)
    s = yield Shuffle(...)
    r = yield Reduce(...)

  def finalized(self):
    if not self.was_aborted:
      blobstore.delete(*self.args)
      self.send_result_email()
```

Production -- Python

```
class MyPipeline(pipeline.Pipeline):
  def run(self, x):
    now = time.time()
    if now % x == 0:
      raise pipeline.Abort('Too low')
    if now % x > 10:
      raise pipeline.Retry('Too high')
    self.set_status(message='Just right')

# From the outside:
job = MyPipeline.from_id(...)
job.abort('forcing fail')
job.retry('forcing retry')
```

Language differences

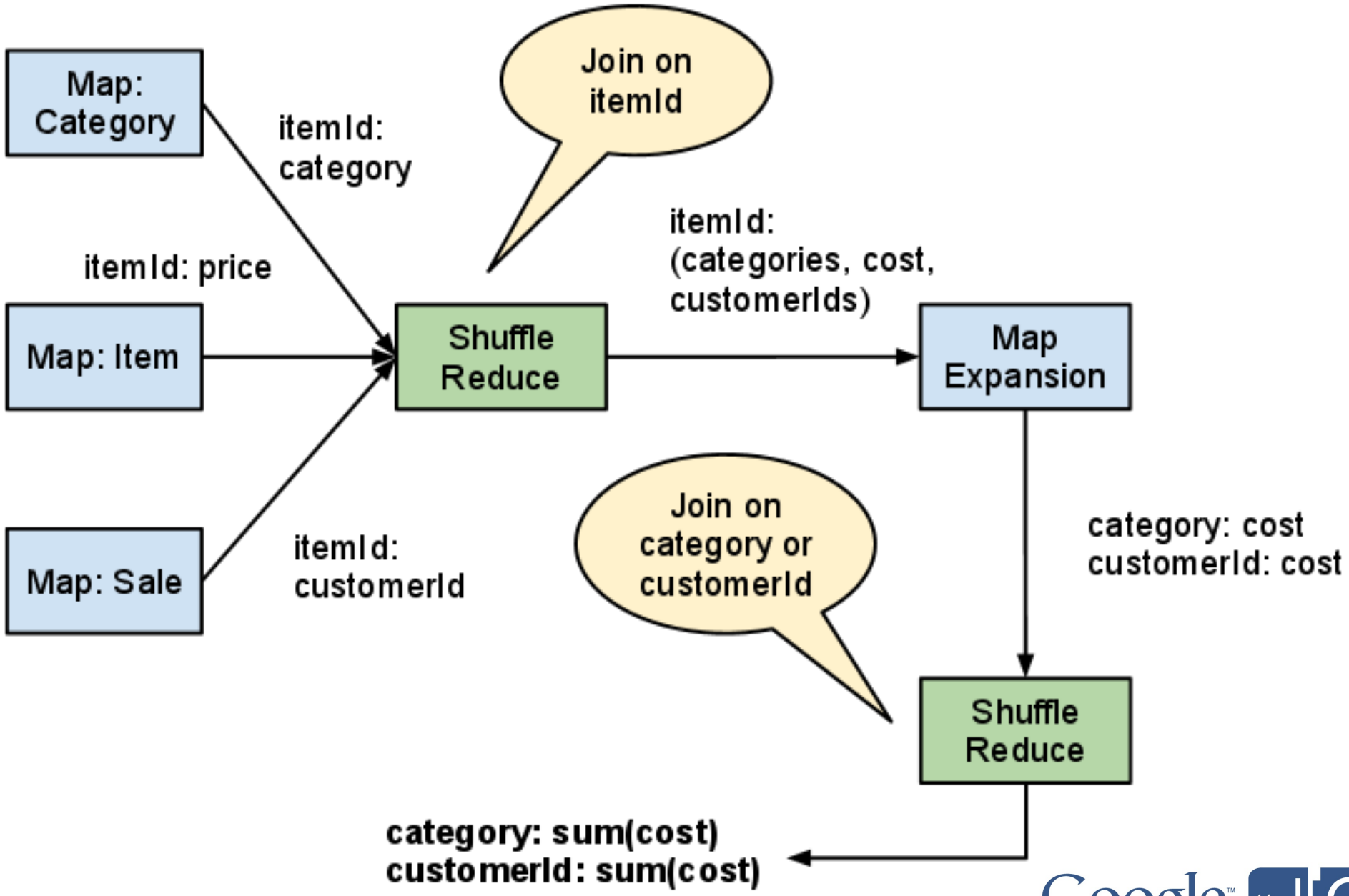
- Coming to Java
  - Functional testing
  - Production optimization
  - Status, links
  - Integration with Mapper Framework
  - (maybe) Finalized

- Not adding to Java
  - Named return values

- Not adding to Python
  - Arbitrary serialized types
  - Promises

Data join example

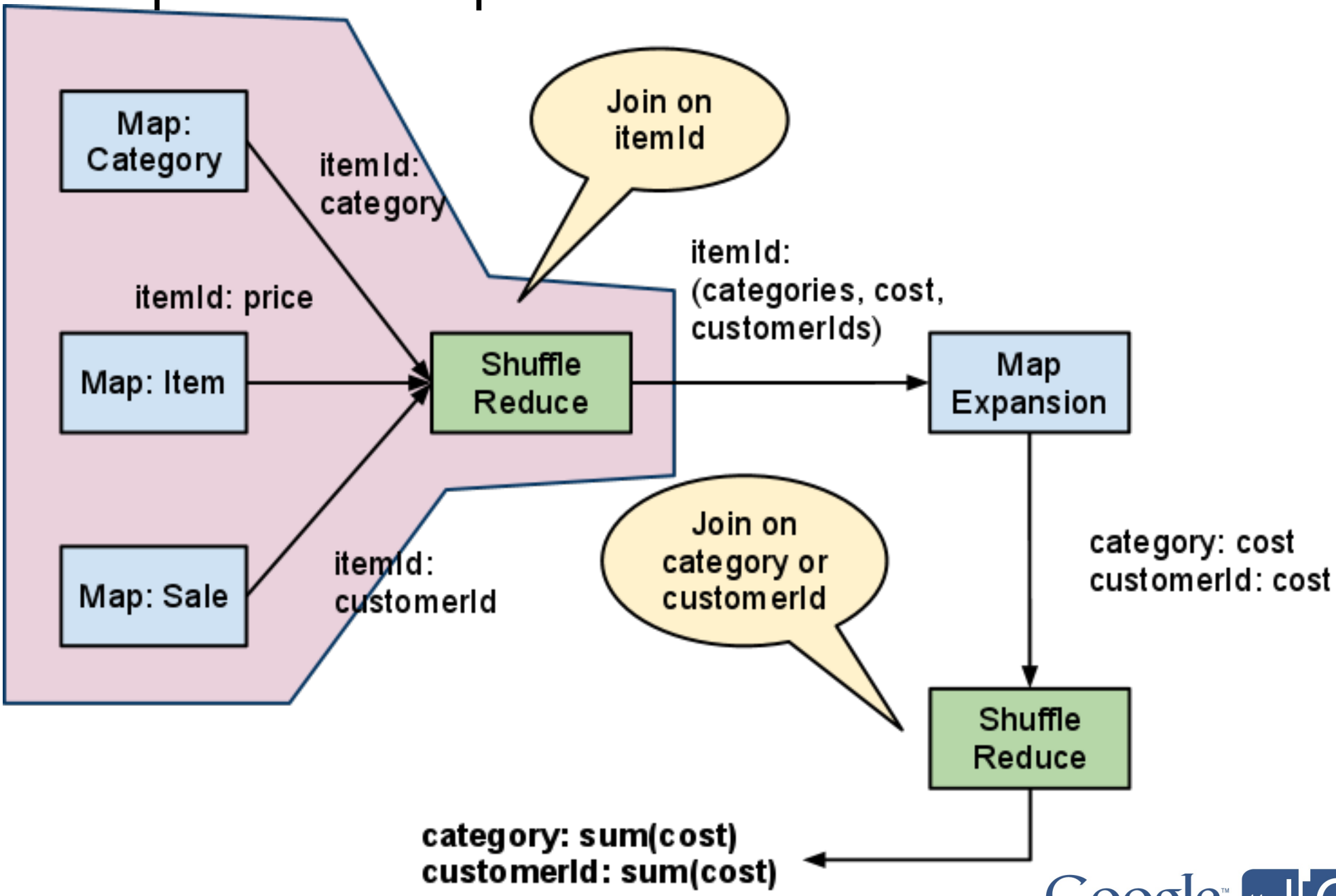
Example: Sales report



Data join via the Pipeline API

```
class JoinOnItemId(pipeline.Pipeline):
  def run(self):
    category_map = yield MapPipeline('cm',
        'map_categories',
        'DatastoreInputReader',
        params=dict(entity_kind='Category'))
    item_map = yield MapPipeline('im',
        'map_items',
        'DatastoreInputReader',
        params=dict(entity_kind='Item'))
    sales_map = yield MapPipeline('sm',
        'map_sales',
        'DatastoreInputReader',
        params=dict(entity_kind='Sales'))
    yield ShufflePipeline((yield Append(
        category_map, item_map, sales_map)))
```

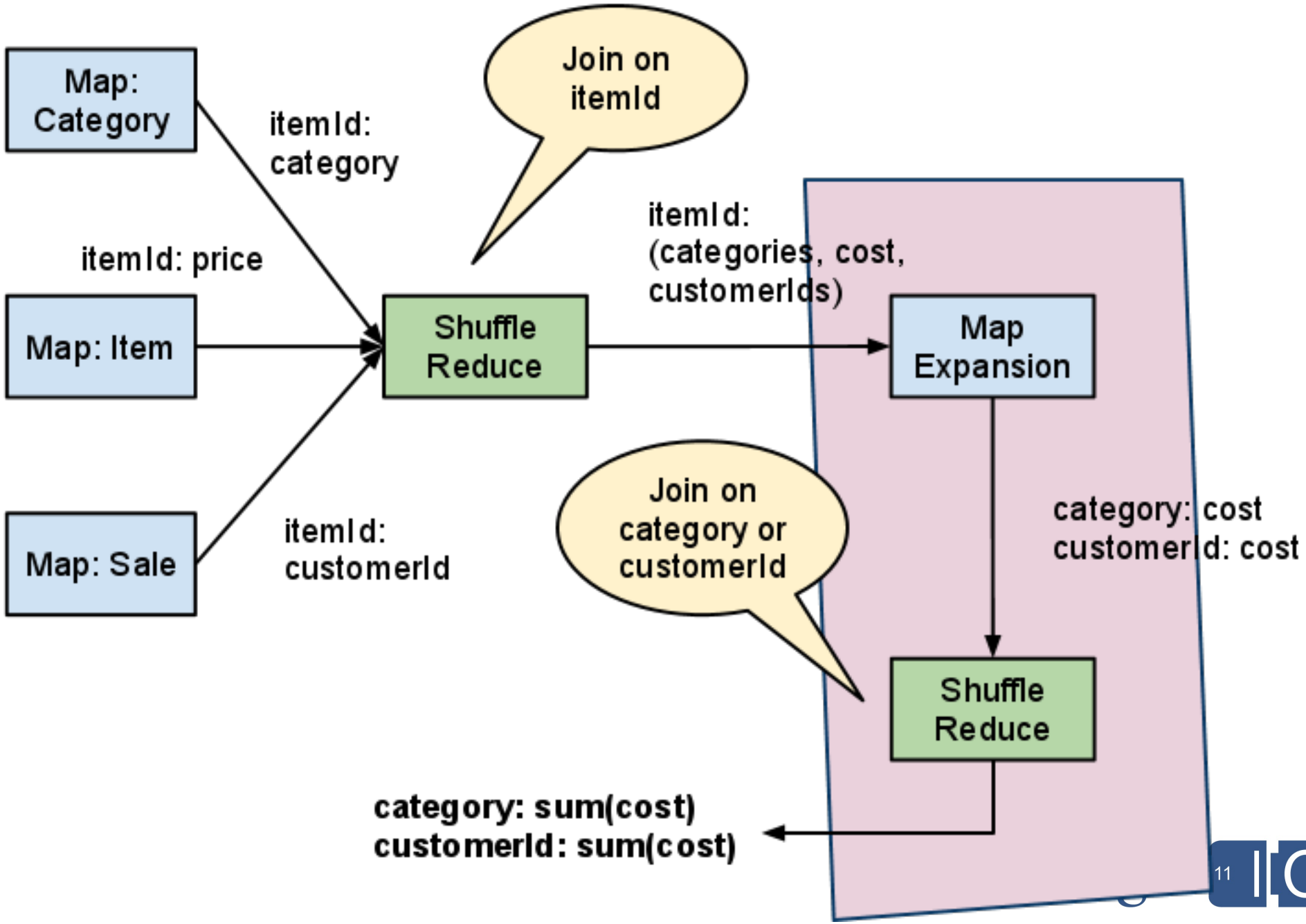
Example: Sales report



Data join via the Pipeline API

```
class SalesReport(pipeline.Pipeline):
  def run(self):
    joined_data = yield JoinOnItemId()
    reduce = yield ReducePipeline('reduce',
        'reduce_items',
        'BlobstoreRecordsOutputWriter',
        params=dict(), joined_data)
    yield MapReducePipeline('phase2',
        'invert_pairs', 'sum_values',
        'RecordsReader',
        'BlobstoreRecordsOutputWriter',
        mapper_params=(yield Dict(files=reduce)))
```

Example: Sales report



Questions ?

Projects:

appengine-pipeline.googlecode.com

mapreduce.appspot.com

Hashtags: `#io2011 #AppEngine`

Feedback: `http://goo.gl/1gGbS`


Google™
11
I/O