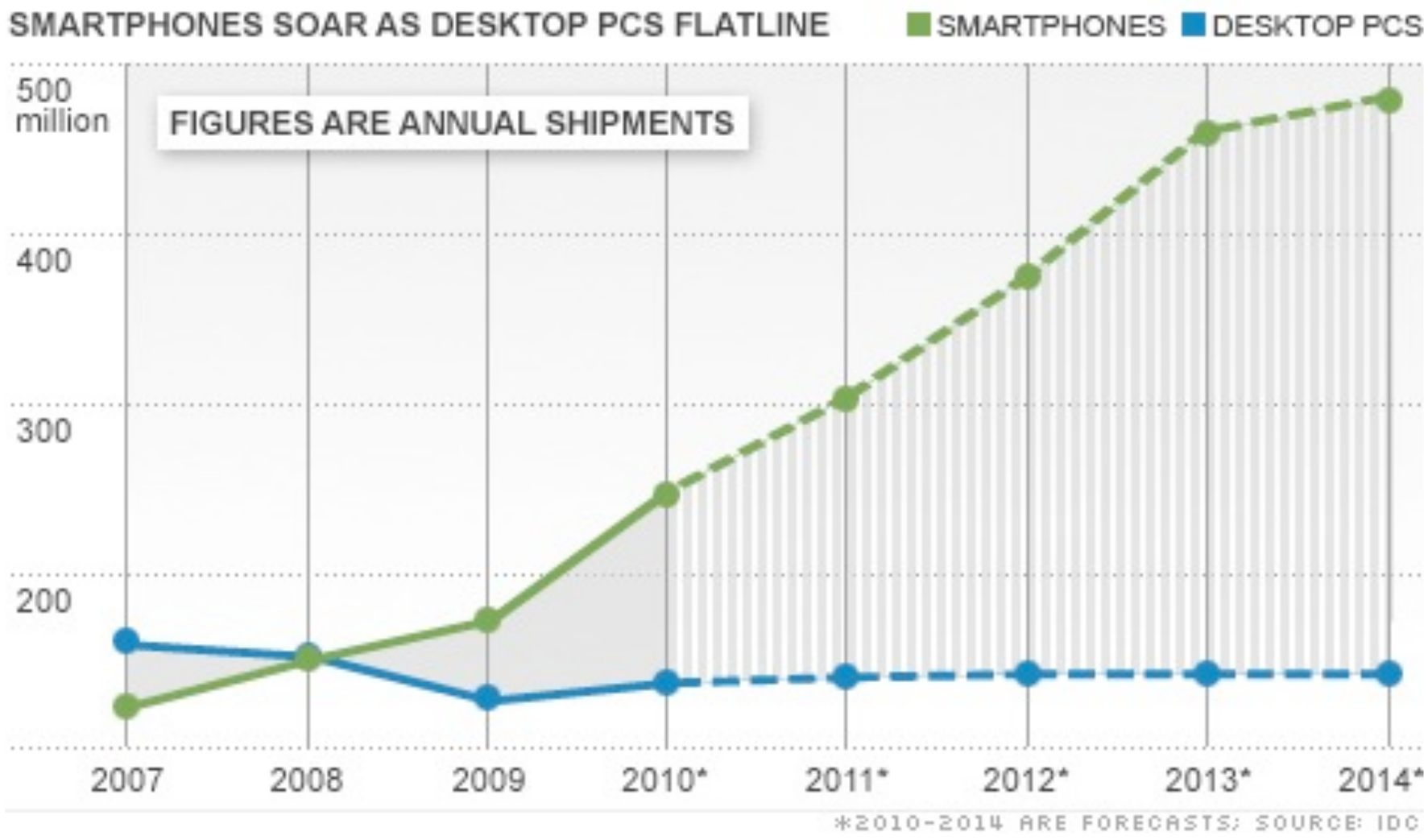
Distributed Version Control for Application Data

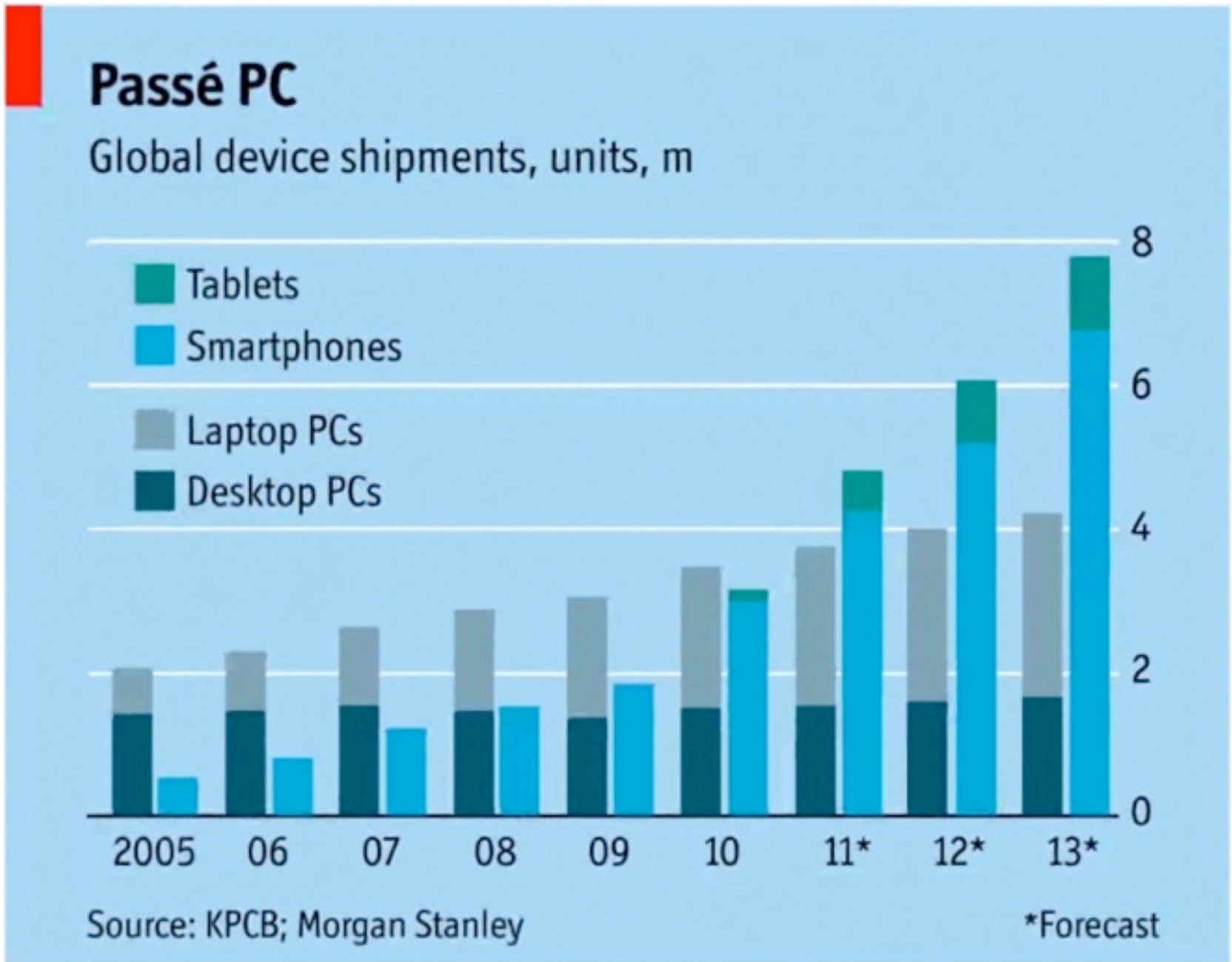
Juan Batiz-Benet
jbenet@cs.stanford.edu

Mobile vs PC

Mobile vs PC

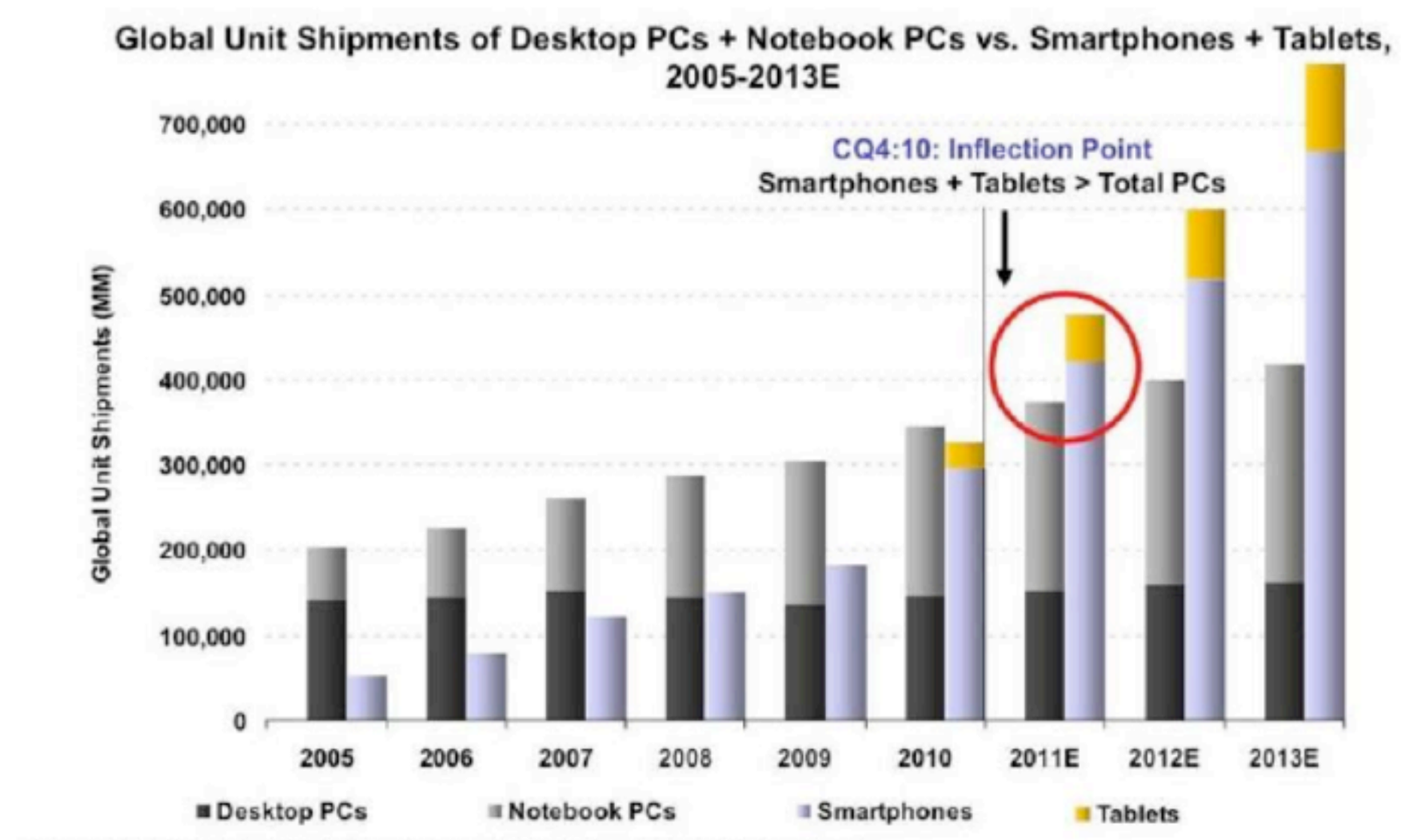

SMARTPHONES SOAR AS DESKTOP PCS FLATLINE
SMARTPHONES
DESKTOP PCS
FIGURES ARE ANNUAL SHIPMENTS
500 million
400
300
200
2007
2008
2009
2010*
2011*
2012*
2013*
2014*
*2010-2014 ARE FORECASTS; SOURCE: IDC

Mobile vs PC


Passé PC
Global device shipments, units, m
Tablets
Smartphones
Laptop PCs
Desktop PCs
8
6
4
2
0
2005
06
07
08
09
10
11*
12*
13*
Source: KPCB; Morgan Stanley
*Forecast

Mobile vs PC

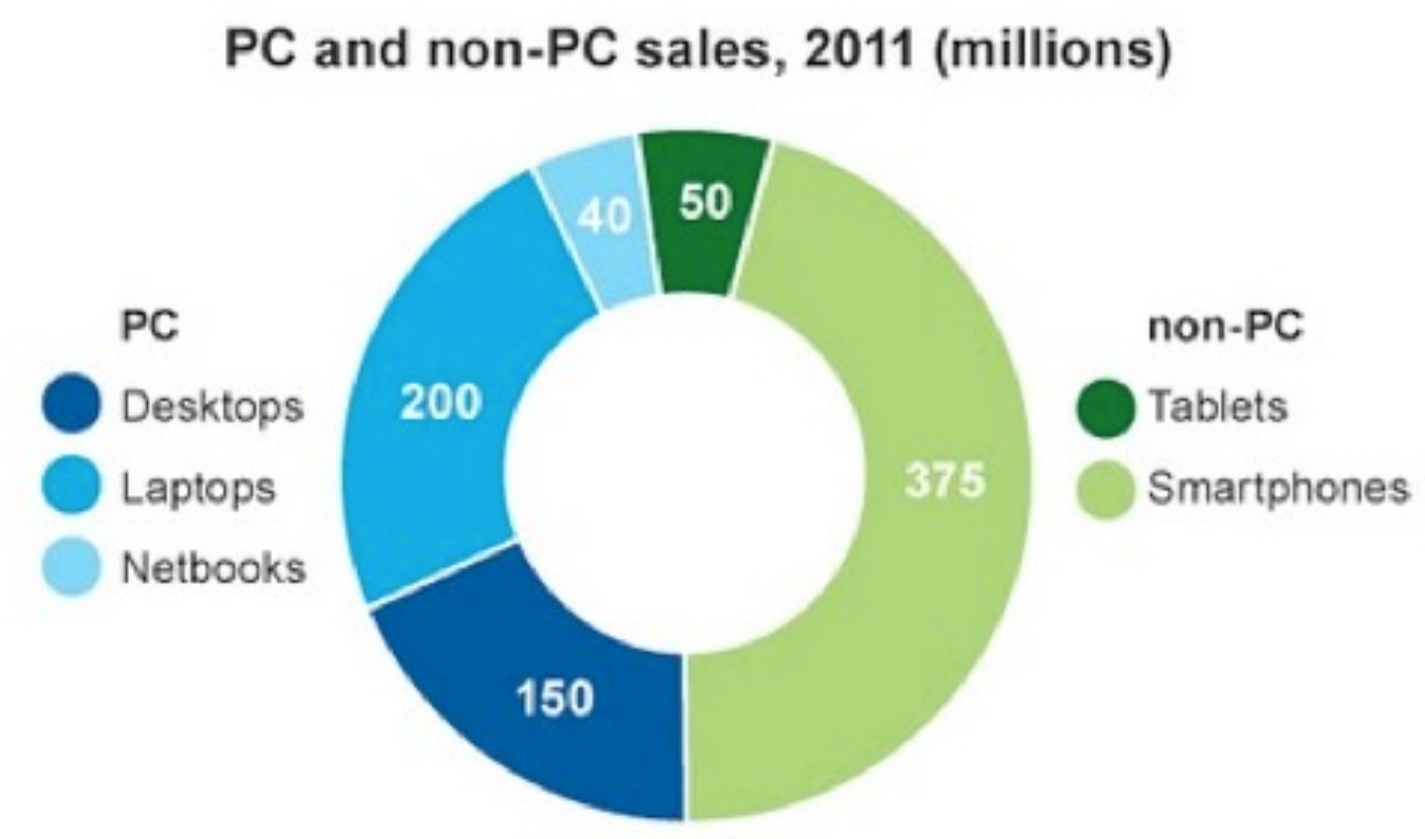
The Dawn of the Post-PC Era



Note: Notebook PCs include Netbooks. Source: Katy Huberty, Ehud Gelblum, Morgan Stanley Research.
Data and Estimates as of 2/11

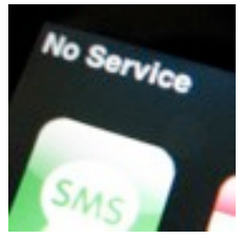
Thanks to Mary Meeker and Matt Murphy

Mobile vs PC


PC and non-PC sales, 2011 (millions)
PC
Desktops
Laptops
Netbooks
200
40
150
50
375
non-PC
Tablets
Smartphones

Distributed Application Data

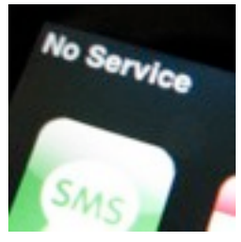
- use apps offline



- merge changes

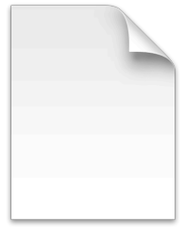
- sync seamlessly

Mobile REQUIRES Distributed Application Data

- use apps offline 

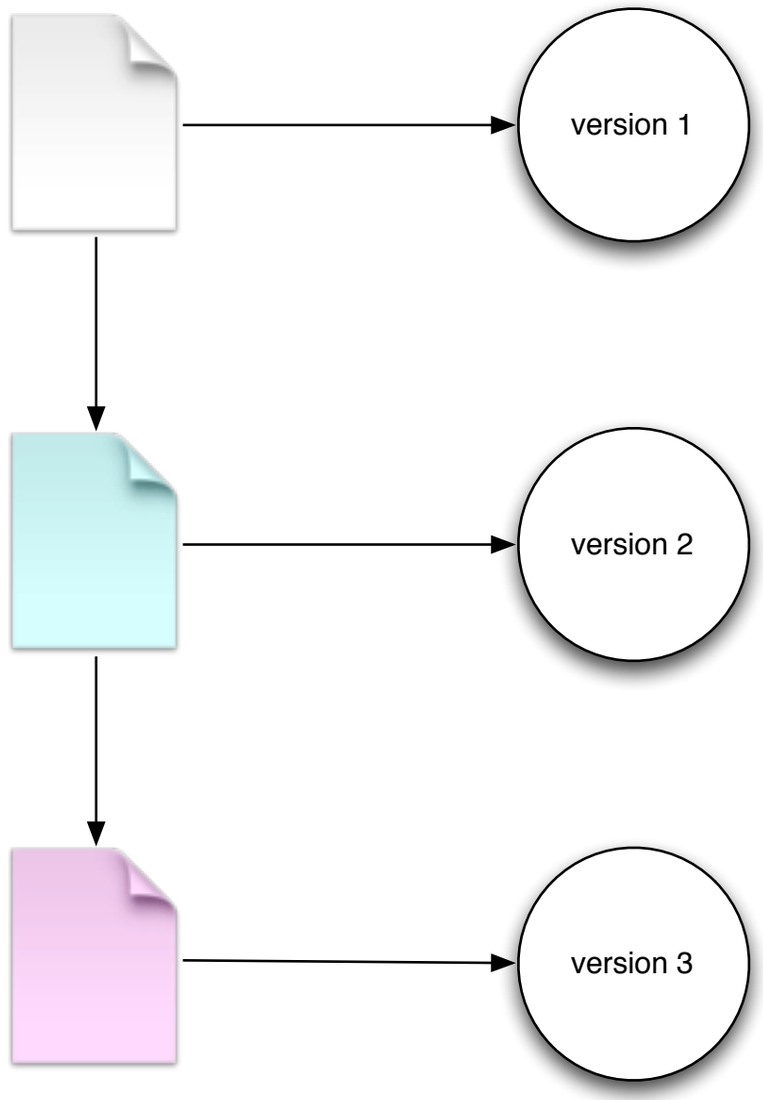
disconnected world
- merge changes  multiple devices
- sync seamlessly  no resolve conflicts

Version Control

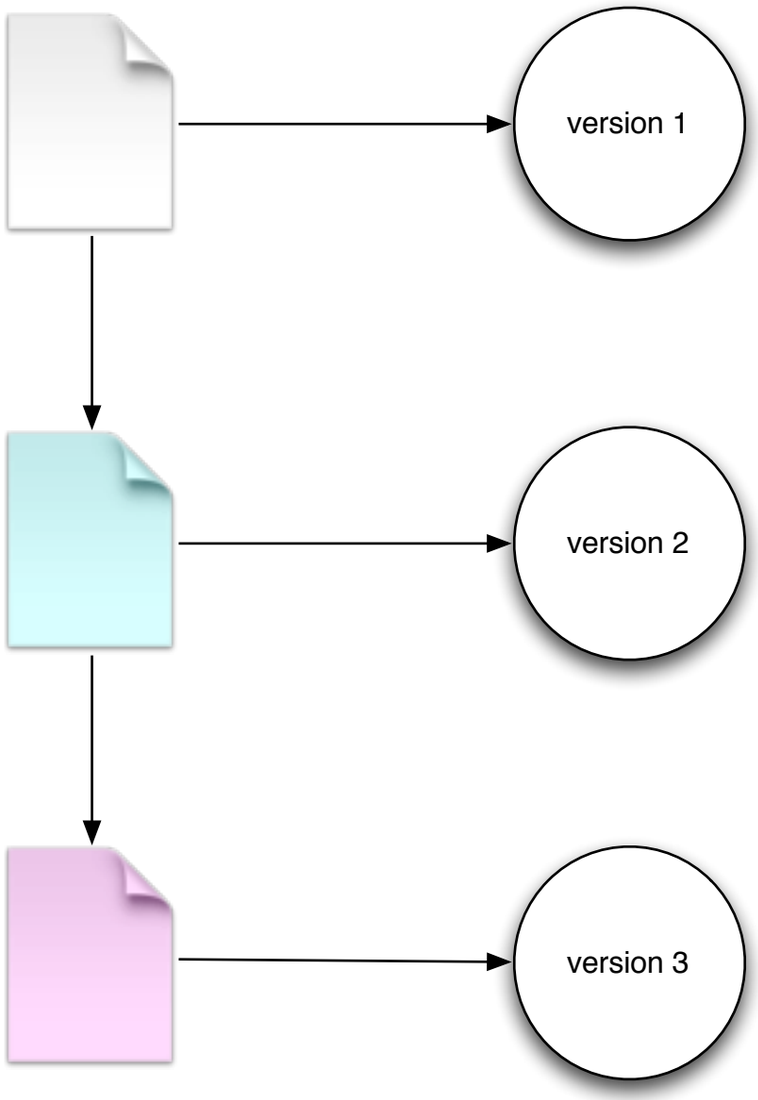


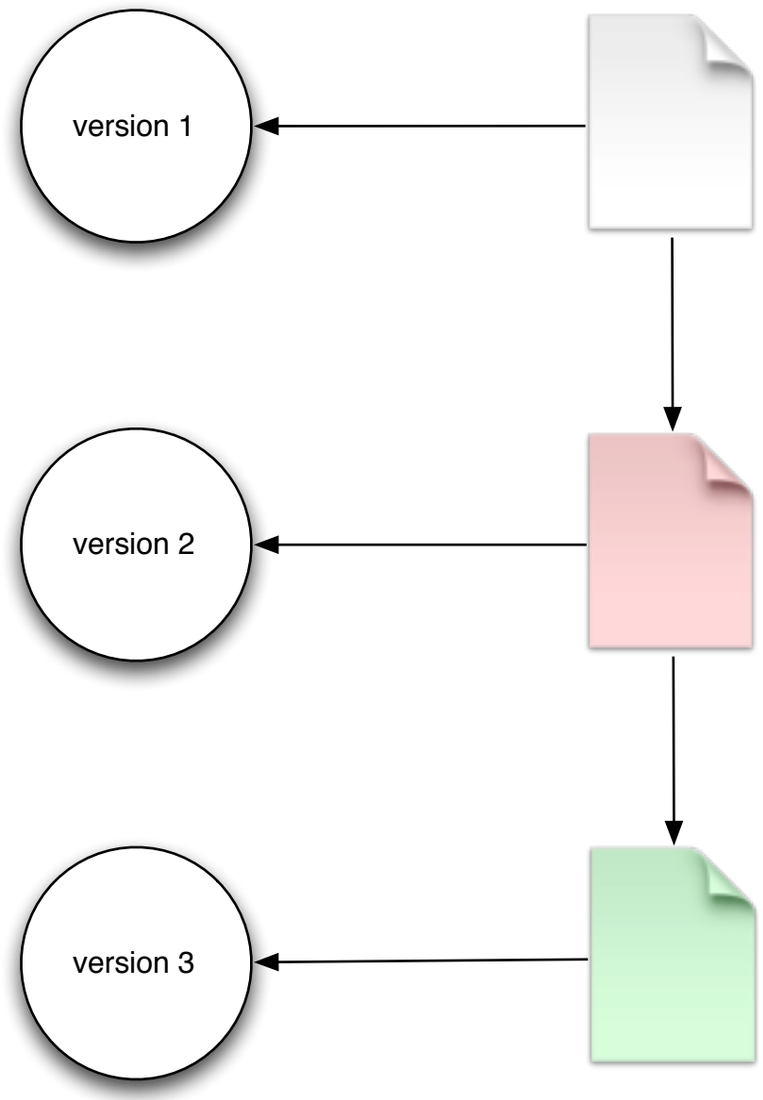
Version Control

Version Control

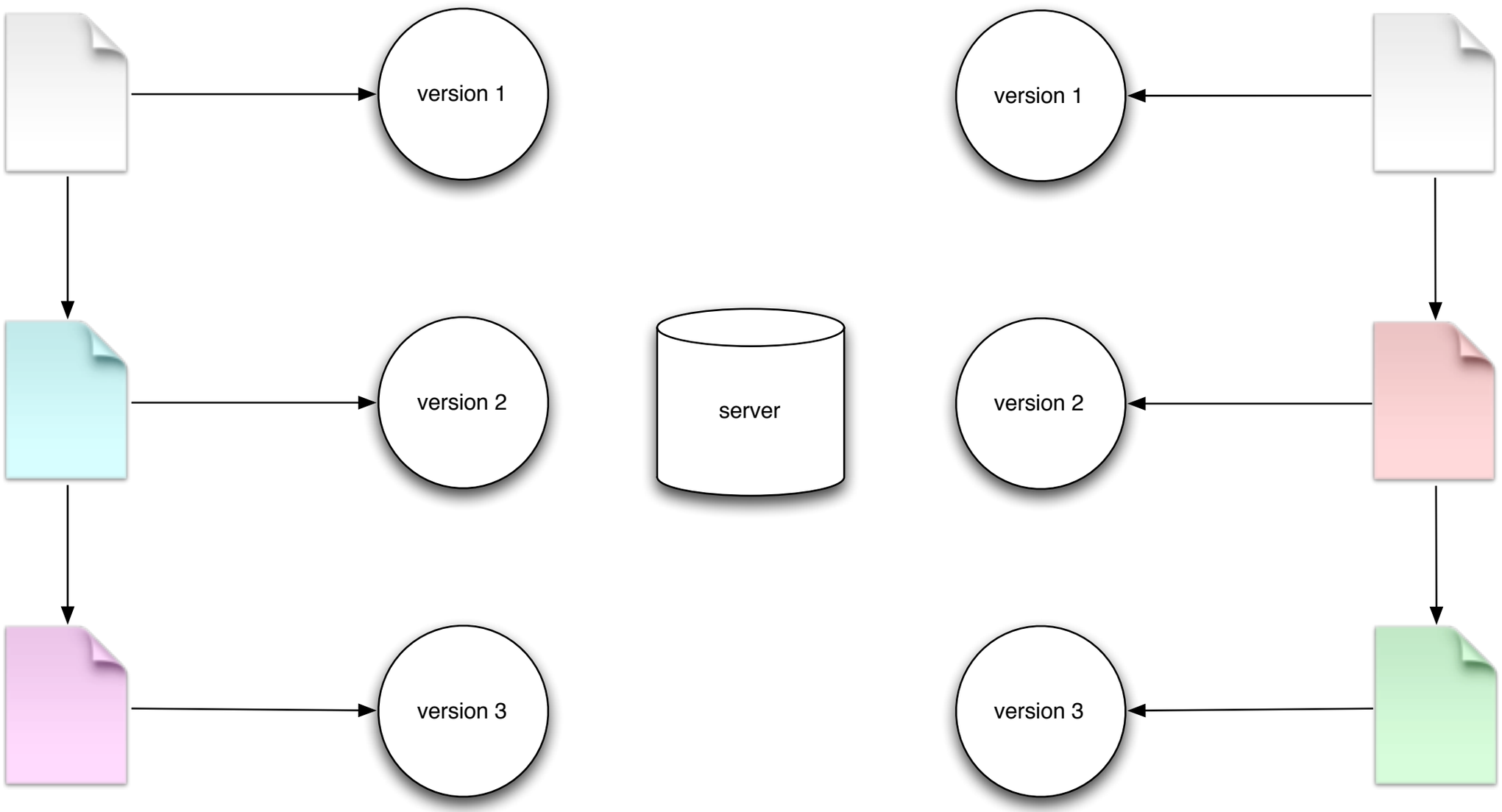


Version Control

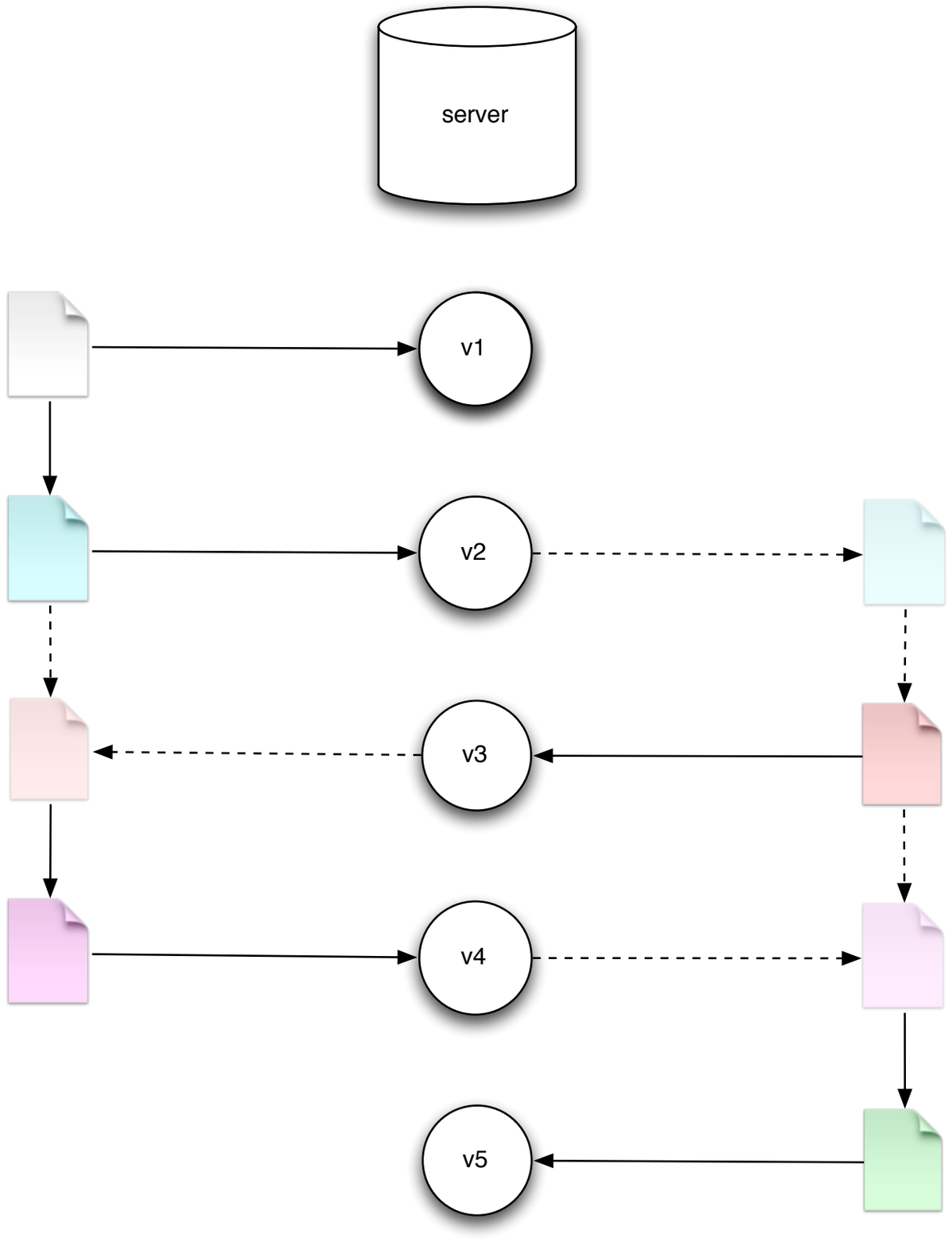




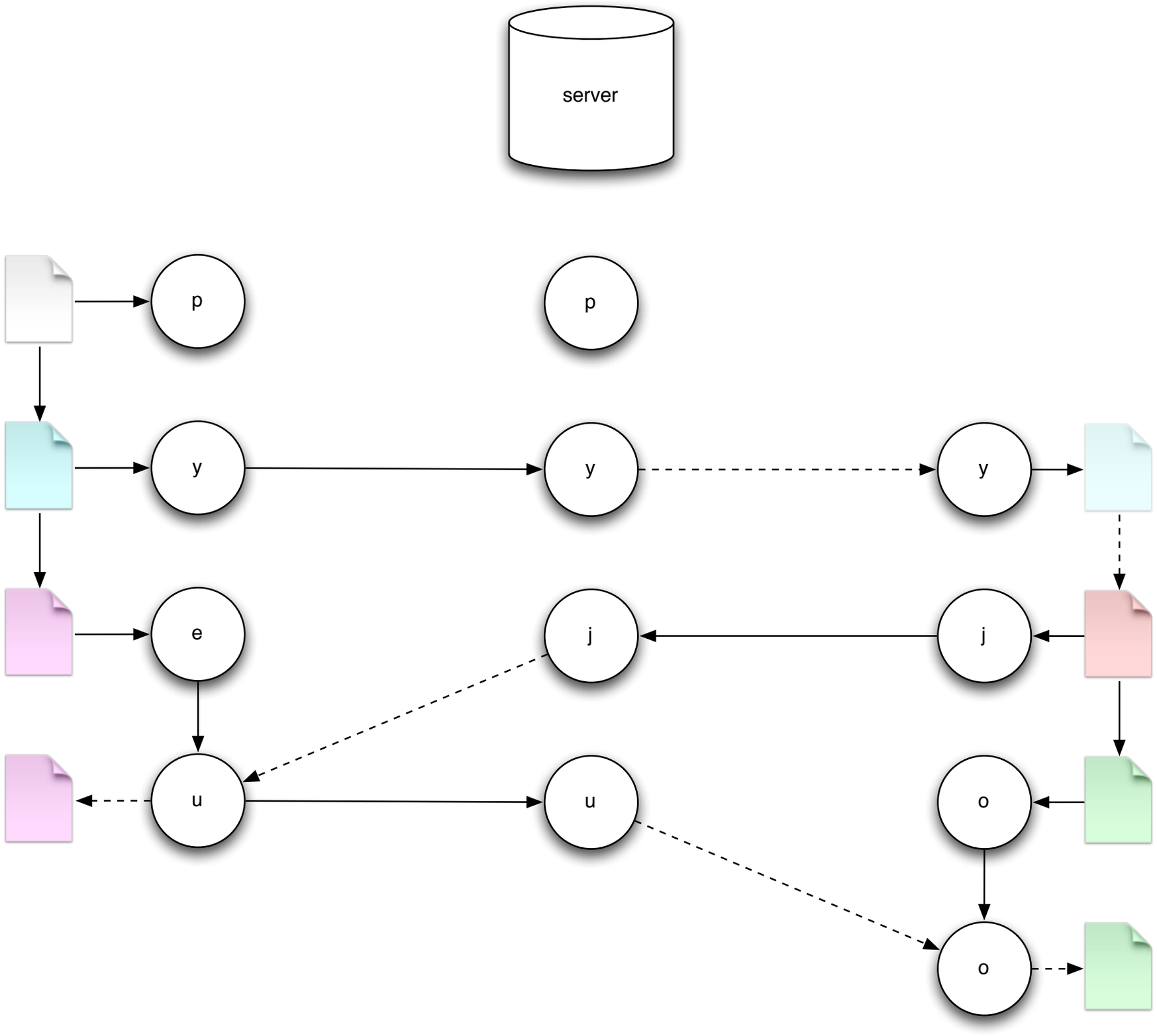
Centralized Version Control



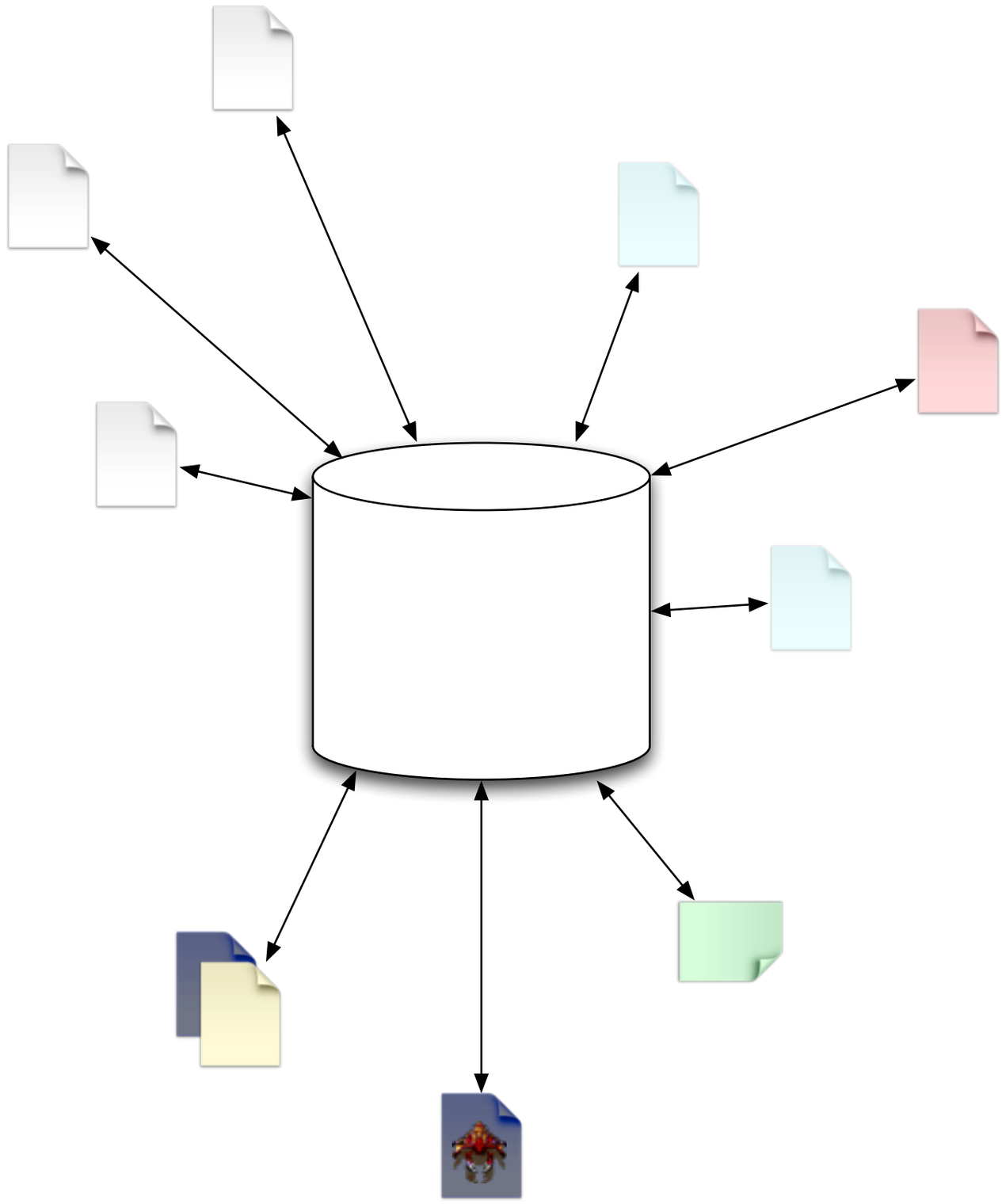
Centralized Version Control



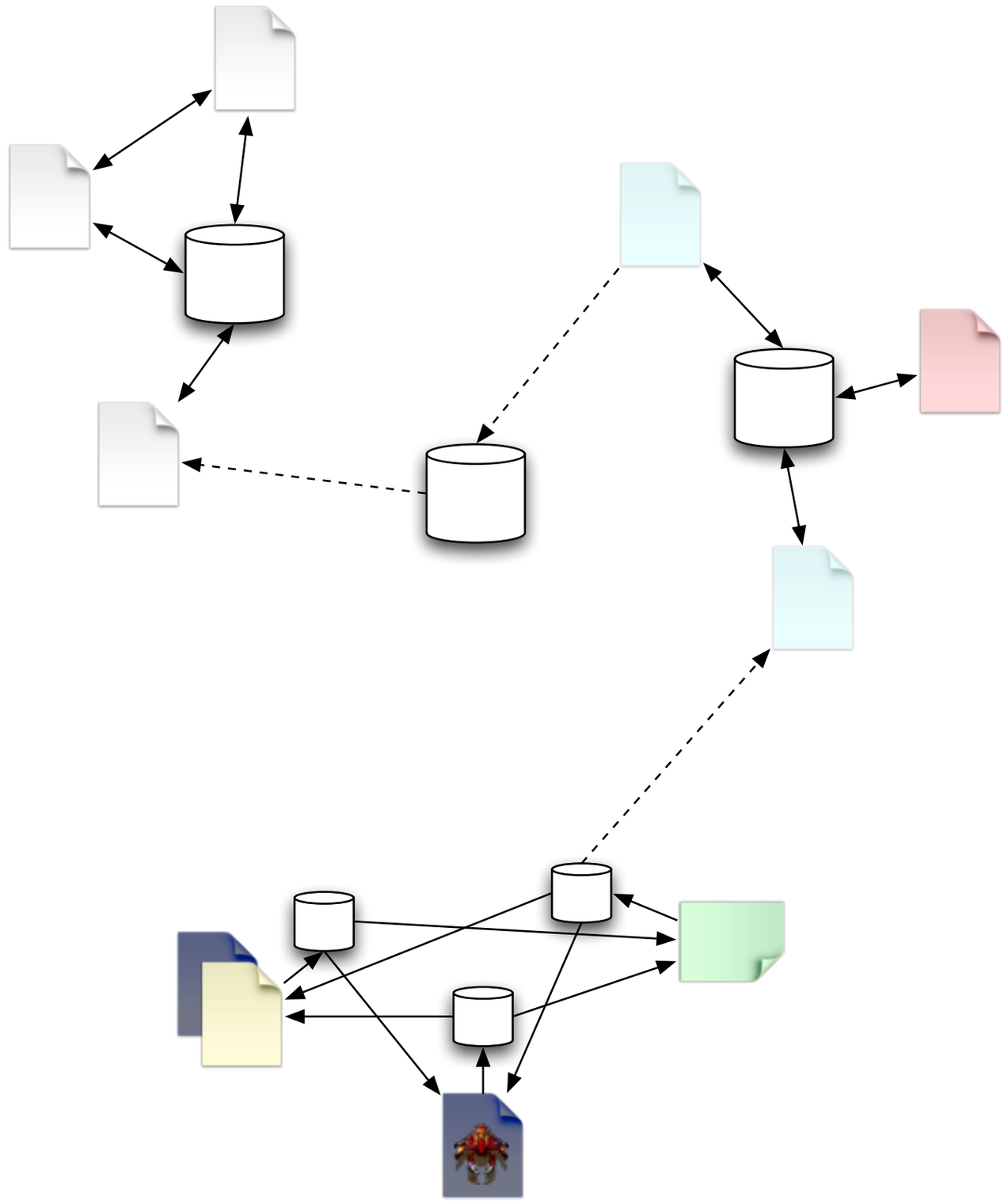
Distributed Version Control



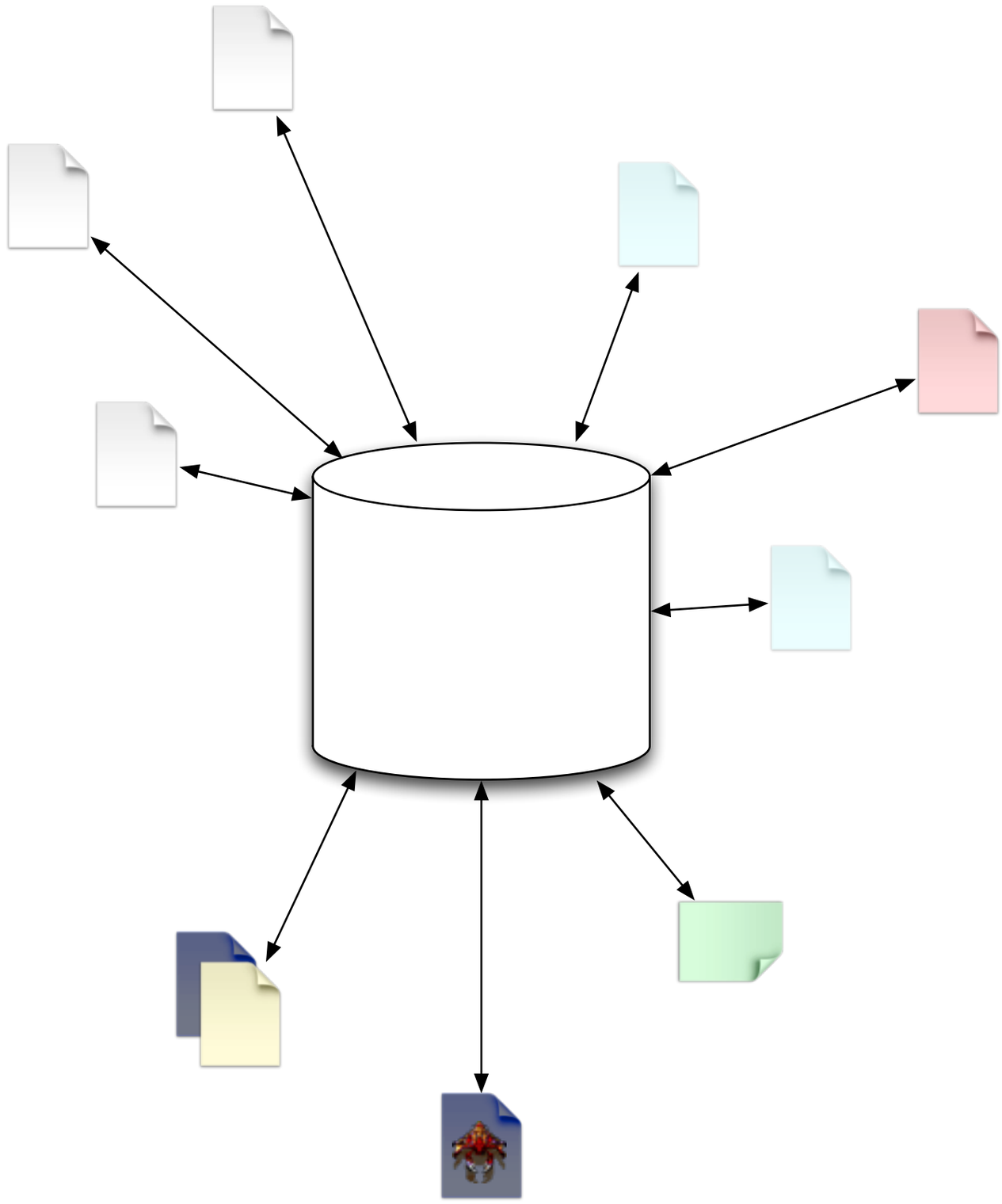
Centralized Version Control



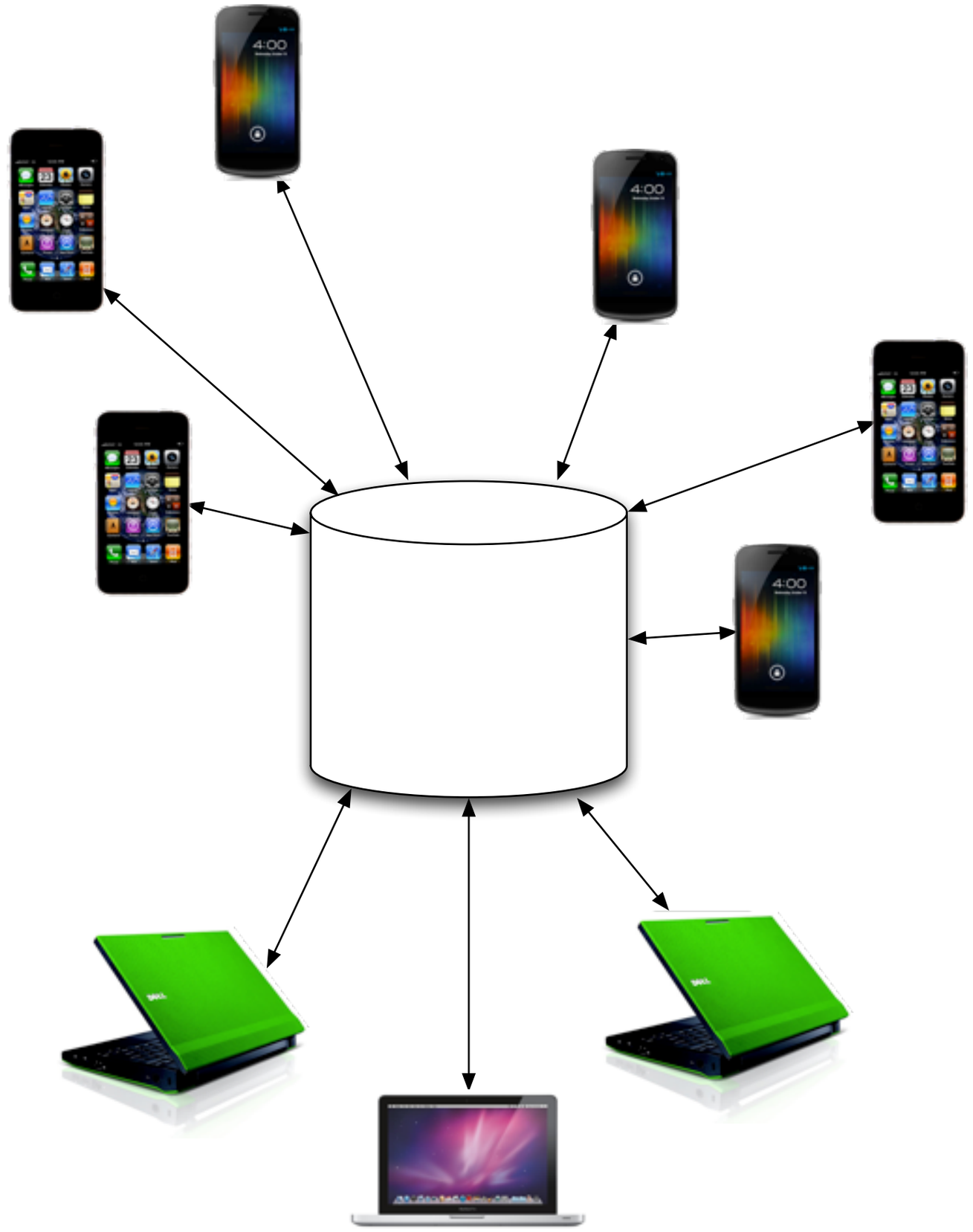
Distributed Version Control



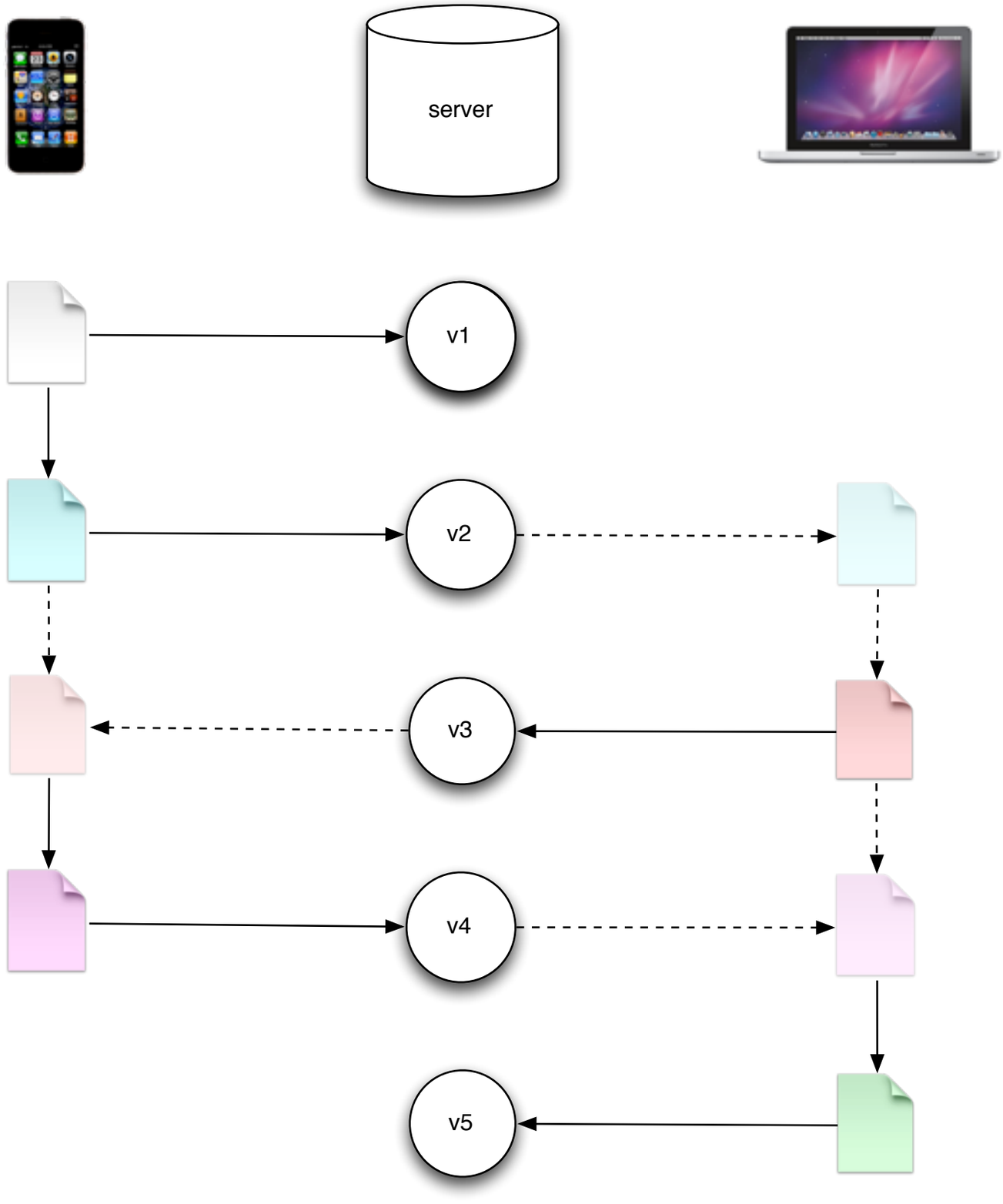
Centralized Version Control



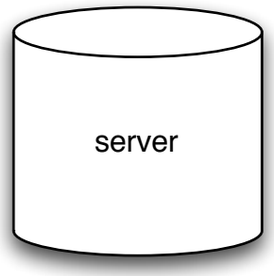
Centralized Application Data

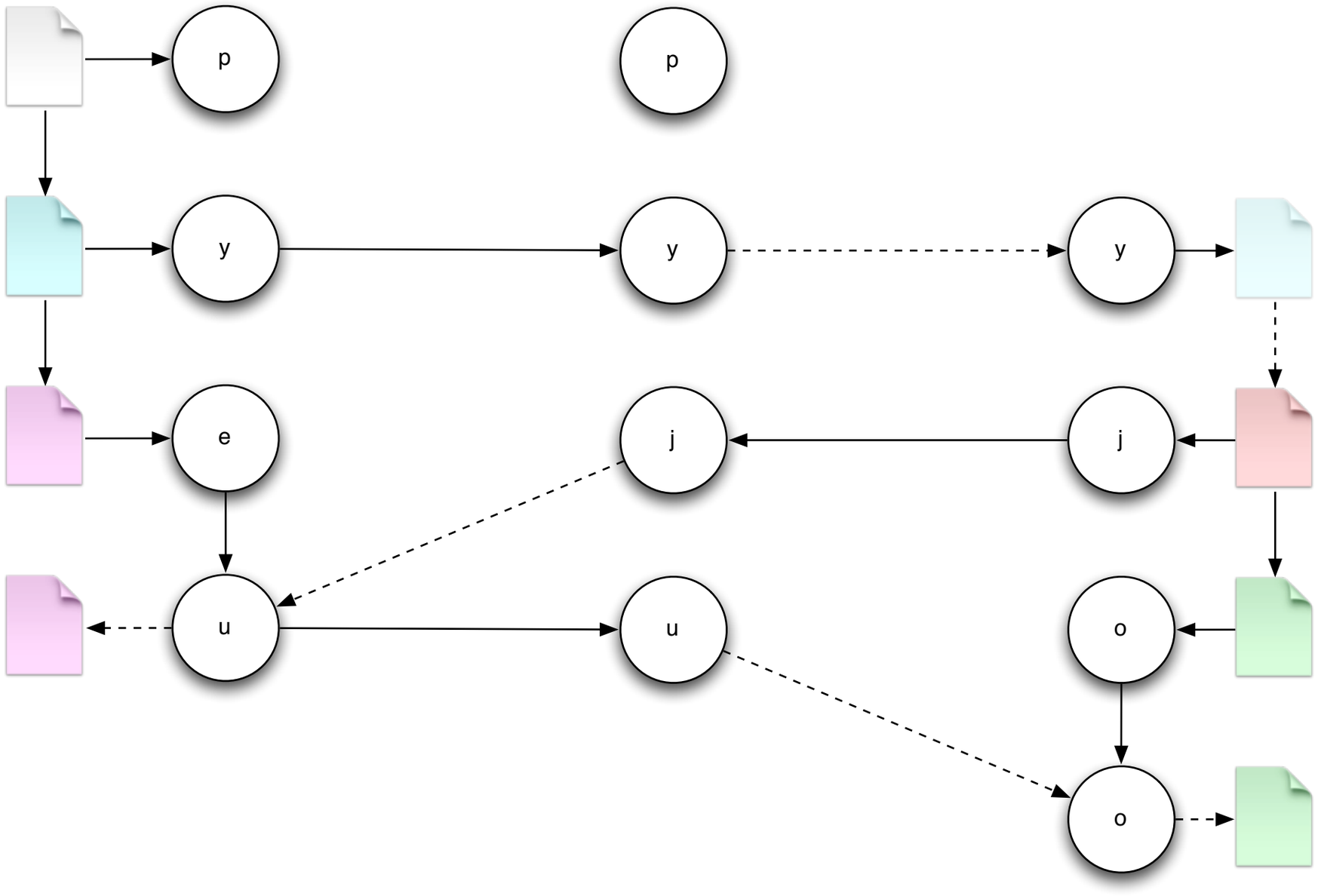


Centralized Application Data



Distributed Application Data?





git in our apps?

git in our apps?

Pros

- sophisticated VC
- tried and true
- core c libs
- py, ruby libs

git in our apps?

Pros

- sophisticated VC
- tried and true
- core c libs
- py, ruby libs

Cons

- on filesystem
- not fast enough
- no data model
- merge conflicts

git in our apps?

Pros

- sophisticated VC
- tried and true
- core c libs
- py, ruby libs

Cons

- on filesystem
- not fast enough
- no data model
- merge conflicts

- store entire tree

git in our apps?

Pros

- sophisticated VC
- tried and true
- core c libs
- py, ruby libs

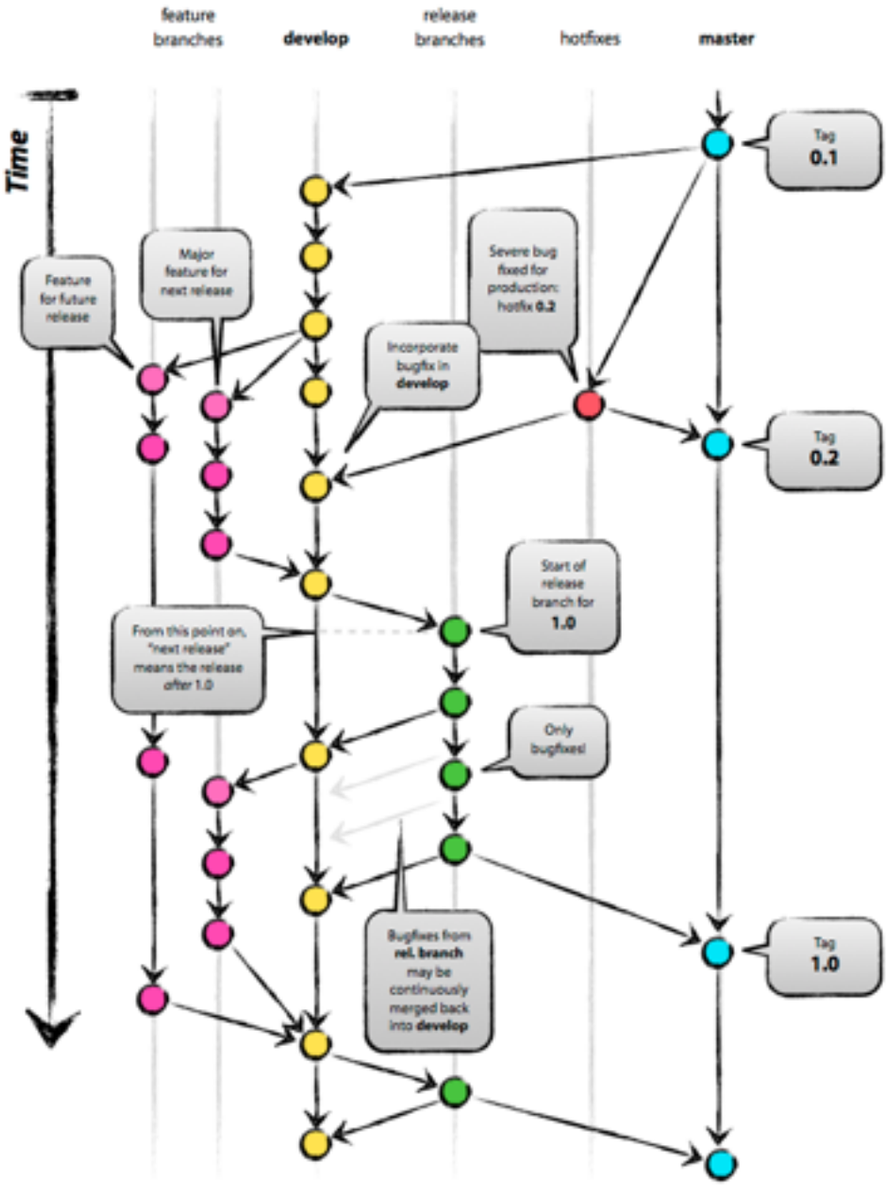
Cons

- on filesystem
- not fast enough
- no data model
- merge conflicts

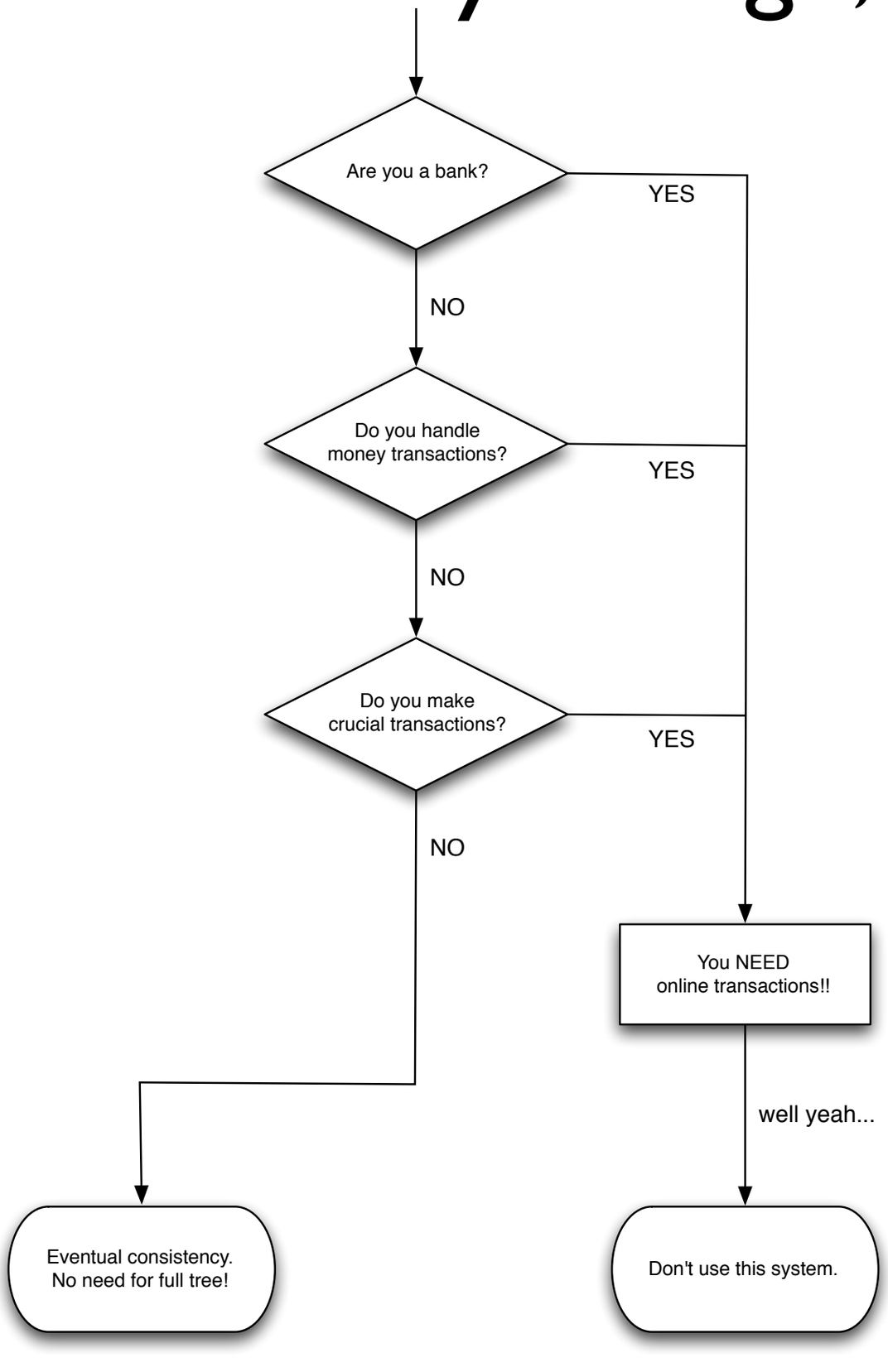
decision

don't use git, build own system

store the entire tree of changes?



do we need every change, ever?



do we need every change, ever?

$$\frac{1 \text{ commit}}{\text{second}} \times \frac{3600 \text{ seconds}}{\text{hour}} \times \frac{11 \text{ hours}}{\text{play day}} = \frac{79200 \text{ commits}}{\text{play day}}$$

per user

how can we keep tree size down?

- commit changes more selectively?
- coalesce commits not externalized?
- regularly prune history?

what data do we really care about?

- most up-to-date object values available

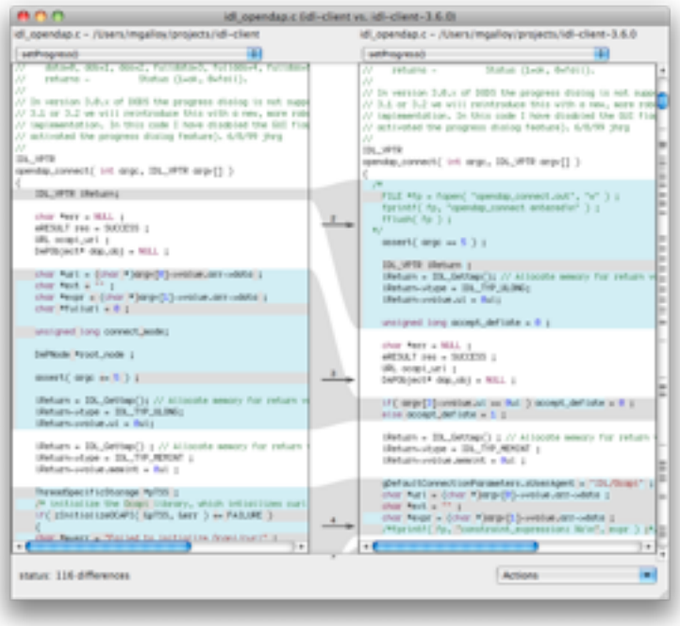
decision

don’t need to keep entire tree

how can we merge objects?

how can we merge objects?

- standard vcs merge algorithms compare text files using position information
- app objects are not source code.



how can we merge objects?

- app objects are collections of attributes

  { attrA: valA, attrB: valB, attrC : valC }

how can we merge objects?

- app objects are collections of attributes

  { attrA: valA, attrB: valB, attrC : valC }

- attrs have clear app-specific semantics

  ‘lastLocation,’ ‘createdAt,’ ‘allVisitors’

how can we merge objects?

- app objects are collections of attributes

  { attrA: valA, attrB: valB, attrC : valC }

- attrs have clear app-specific semantics

  ‘lastLocation,’ ‘createdAt,’ ‘allVisitors’

- merging must capture those semantics

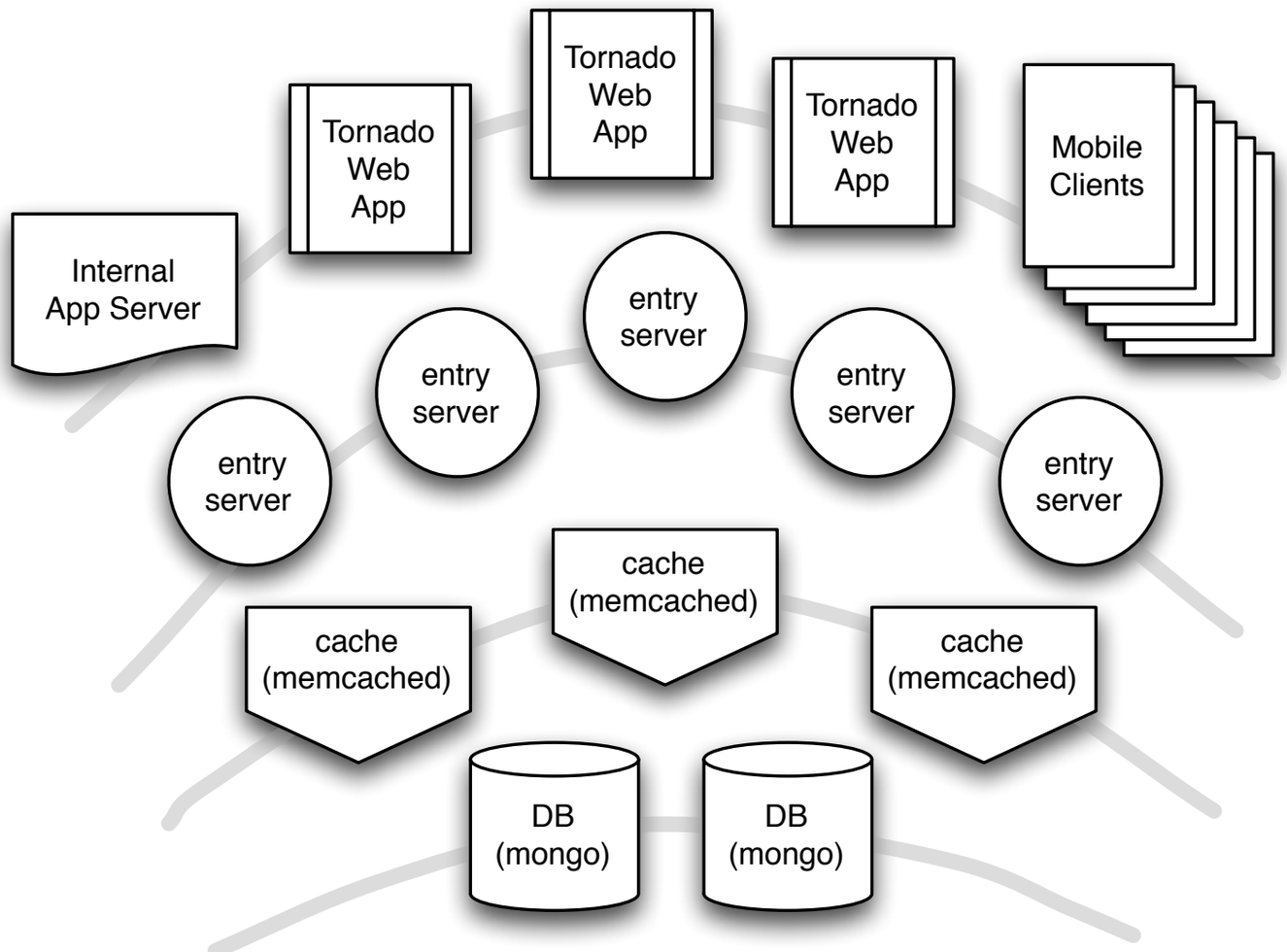
```
def merge(loc1, loc2):
  '''return the last updated location'''
  if loc2.updated > loc1.updated:
    return loc2
  return loc1
```

how can we merge objects?

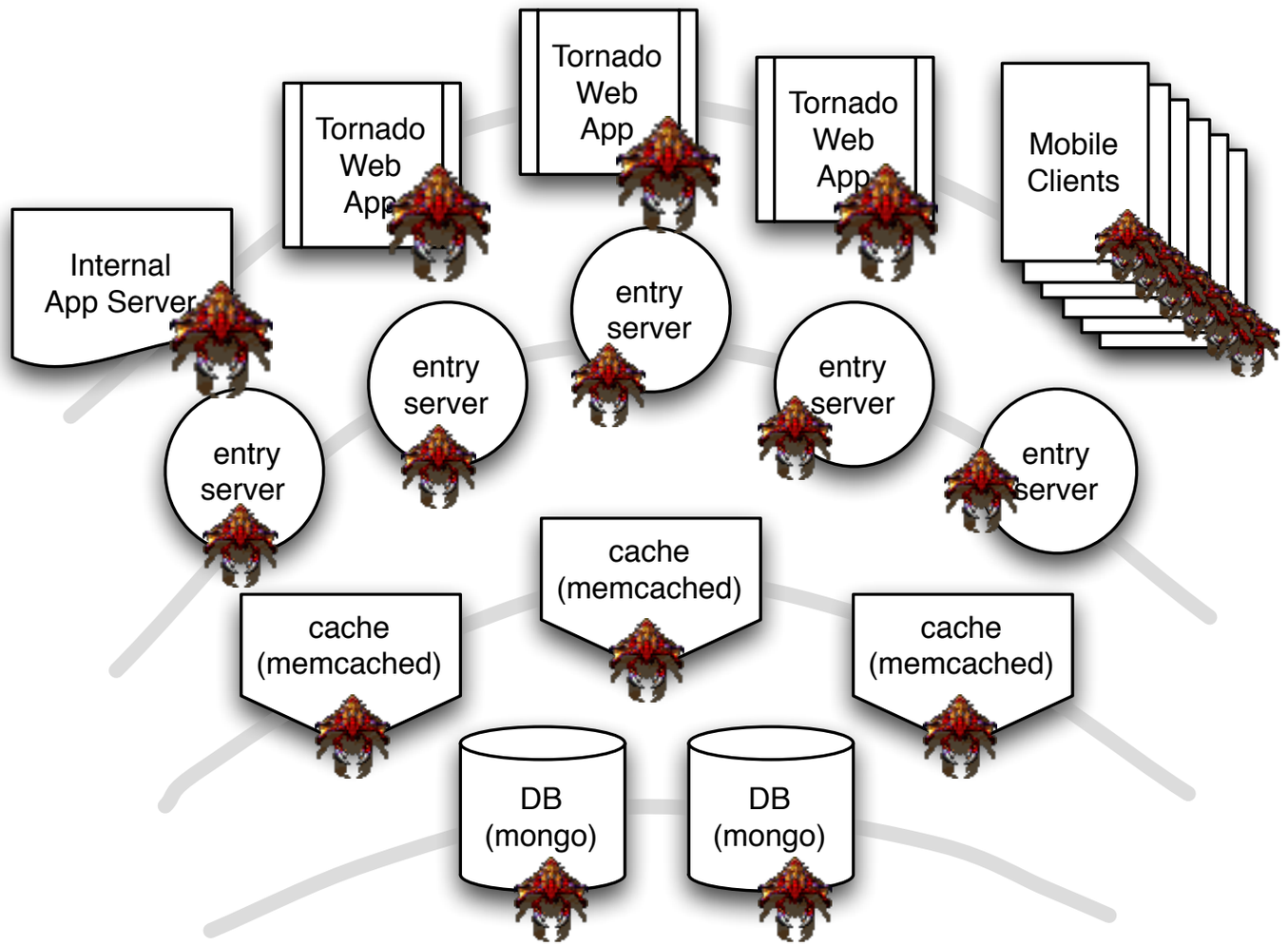
decision

build own merging scheme that lets programmers specify what and how to merge, on a per-attribute basis

DroneStore


Tornado
Web
App
Tornado
Web
App
Tornado
Web
App
Mobile
Clients
Internal
App Server
entry
server
entry
server
entry
server
entry
server
entry
server
cache
(memcached)
cache
(memcached)
cache
(memcached)
DB
(mongo)
DB
(mongo)

DroneStore


Tornado
Web
App
Tornado
Web
App
Tornado
Web
App
Mobile
Clients
Internal
App Server
entry
server
entry
server
entry
server
entry
server
entry
server
cache
(memcached)
cache
(memcached)
cache
(memcached)
DB
(mongo)
DB
(mongo)

History

- two years ago: original idea
- fall 2010: py and obj-c libs v0
- winter 2011: c++ lib for cs249b
- June 2011: py and obj-c libs v1

- running in production! :]

Hello World

```
>>> import dronestore
>>> from dronestore import StringAttribute
>>> from dronestore.merge import LatestStrategy
>>>
>>> class MyModel(dronestore.Model):
...   first = StringAttribute(strategy=LatestStrategy)
...   second = StringAttribute(strategy=LatestStrategy)
...
>>> foo = MyModel('FooBar')
>>> foo.first = 'Hello'
>>> foo.commit()
>>>
>>> bar = MyModel('FooBar')
>>> bar.second = 'World'
>>> bar.commit()
>>>
>>> foo.merge(bar)
>>> print foo.first, foo.second
Hello World
```

Hello World

```
>>> import dronestore
>>> from dronestore import StringAttribute
>>> from dronestore.merge import LatestStrategy
>>>
>>> class MyModel(dronestore.Model):
...   first = StringAttribute(strategy=LatestStrategy)
...   second = StringAttribute(strategy=LatestStrategy)
...
>>> foo = MyModel('FooBar')
>>> foo.first = 'Hello'
>>> foo.commit()
>>>
>>> bar = MyModel('FooBar')
>>> bar.second = 'World'
>>> bar.commit()
>>>
>>> foo.merge(bar)
>>> print foo.first, foo.second
Hello World
```

Hello World

```
>>> import dronestore
>>> from dronestore import StringAttribute
>>> from dronestore.merge import LatestStrategy
>>>
>>> class MyModel(dronestore.Model):
...     first = StringAttribute(strategy=LatestStrategy)
...     second = StringAttribute(strategy=LatestStrategy)
...
>>> foo = MyModel('FooBar')
>>> foo.first = 'Hello'
>>> foo.commit()
>>>
>>> bar = MyModel('FooBar')
>>> bar.second = 'World'
>>> bar.commit()
>>>
>>> foo.merge(bar)
>>> print foo.first, foo.second
Hello World
```

Hello World

```
>>> import dronestore
>>> from dronestore import StringAttribute
>>> from dronestore.merge import LatestStrategy
>>>
>>> class MyModel(dronestore.Model):
...   first = StringAttribute(strategy=LatestStrategy)
...   second = StringAttribute(strategy=LatestStrategy)
...
>>> foo = MyModel('FooBar')
>>> foo.first = 'Hello'
>>> foo.commit()
>>>
>>> bar = MyModel('FooBar')
>>> bar.second = 'World'
>>> bar.commit()
>>>
>>> foo.merge(bar)
>>> print foo.first, foo.second
Hello World
```

Hello World

```
>>> import dronestore
>>> from dronestore import StringAttribute
>>> from dronestore.merge import LatestStrategy
>>>
>>> class MyModel(dronestore.Model):
...   first = StringAttribute(strategy=LatestStrategy)
...   second = StringAttribute(strategy=LatestStrategy)
...
>>> foo = MyModel('FooBar')
>>> foo.first = 'Hello'
>>> foo.commit()
>>>
>>> bar = MyModel('FooBar')
>>> bar.second = 'World'
>>> bar.commit()
>>>
>>> foo.merge(bar)
>>> print foo.first, foo.second
Hello World
```

Hello World

before merge

```
>>> foo.version.serialRepresentation.data()
{
    "attributes": {
        "first": {
            "updated": 1326476344447048960,
            "value": "Hello"
        },
        "second": {
            "updated": 0,
            "value": null
        }
    },
    "committed": 1326476347431011840,
    "created": 1326476347431011840,
    "hash": "8b52f06b5c2a6e5da...57da3a887",
    "key": "/MyModel/FooBar",
    "parent": "000000000000000...000000000",
    "type": "MyModel"
}
```

foo

```
>>> bar.version.serialRepresentation.data()
{
    "attributes": {
        "first": {
            "updated": 0,
            "value": null
        },
        "second": {
            "updated": 1326476356582902016,
            "value": "World"
        }
    },
    "committed": 1326476359982740992,
    "created": 1326476359982740992,
    "hash": "491734079555d64b6...58f63c5d7",
    "key": "/MyModel/FooBar",
    "parent": "000000000000000...000000000",
    "type": "MyModel"
}
```

bar

Hello World

before merge

```
>>> foo.version.serialRepresentation.data()
{
    "attributes": {
        "first": {
            "updated": 1326476344447048960,
            "value": "Hello"
        },
        "second": {
            "updated": 0,
            "value": null
        }
    },
    "committed": 1326476347431011840,
    "created": 1326476347431011840,
    "hash": "8b52f06b5c2a6e5da...57da3a887",
    "key": "/MyModel/FooBar",
    "parent": "000000000000000...000000000",
    "type": "MyModel"
}
```

foo

```
>>> bar.version.serialRepresentation.data()
{
    "attributes": {
        "first": {
            "updated": 0,
            "value": null
        },
        "second": {
            "updated": 1326476356582902016,
            "value": "World"
        }
    },
    "committed": 1326476359982740992,
    "created": 1326476359982740992,
    "hash": "491734079555d64b6...58f63c5d7",
    "key": "/MyModel/FooBar",
    "parent": "000000000000000...000000000",
    "type": "MyModel"
}
```

bar

Hello World

before merge

```
>>> foo.version.serialRepresentation.data()
{
    "attributes": {
        "first": {
            "updated": 1326476344447048960,
            "value": "Hello"
        },
        "second": {
            "updated": 0,
            "value": null
        }
    },
    "committed": 1326476347431011840,
    "created": 1326476347431011840,
    "hash": "8b52f06b5c2a6e5da...57da3a887",
    "key": "/MyModel/FooBar",
    "parent": "0000000000000000...000000000",
    "type": "MyModel"
}
```

foo

```
>>> bar.version.serialRepresentation.data()
{
    "attributes": {
        "first": {
            "updated": 0,
            "value": null
        },
        "second": {
            "updated": 1326476356582902016,
            "value": "World"
        }
    },
    "committed": 1326476359982740992,
    "created": 1326476359982740992,
    "hash": "491734079555d64b6...58f63c5d7",
    "key": "/MyModel/FooBar",
    "parent": "0000000000000000...000000000",
    "type": "MyModel"
}
```

bar

Hello World

before merge

```
>>> foo.version.serialRepresentation.data()
{
    "attributes": {
        "first": {
            "updated": 1326476344447048960,
            "value": "Hello"
        },
        "second": {
            "updated": 0,
            "value": null
        }
    },
    "committed": 1326476347431011840,
    "created": 1326476347431011840,
    "hash": "8b52f06b5c2a6e5da...57da3a887",
    "key": "/MyModel/FooBar",
    "parent": "000000000000000...000000000",
    "type": "MyModel"
}
```

foo

```
>>> bar.version.serialRepresentation.data()
{
    "attributes": {
        "first": {
            "updated": 0,
            "value": null
        },
        "second": {
            "updated": 1326476356582902016,
            "value": "World"
        }
    },
    "committed": 1326476359982740992,
    "created": 1326476359982740992,
    "hash": "491734079555d64b6...58f63c5d7",
    "key": "/MyModel/FooBar",
    "parent": "000000000000000...000000000",
    "type": "MyModel"
}
```

bar

Hello World

before merge

```
>>> foo.version.serialRepresentation.data()
{
    "attributes": {
        "first": {
            "updated": 1326476344447048960,
            "value": "Hello"
        },
        "second": {
            "updated": 0,
            "value": null
        }
    },
    "committed": 1326476347431011840,
    "created": 1326476347431011840,
    "hash": "8b52f06b5c2a6e5da...57da3a887",
    "key": "/MyModel/FooBar",
    "parent": "000000000000000...000000000",
    "type": "MyModel"
}
```

foo

```
>>> bar.version.serialRepresentation.data()
{
    "attributes": {
        "first": {
            "updated": 0,
            "value": null
        },
        "second": {
            "updated": 1326476356582902016,
            "value": "World"
        }
    },
    "committed": 1326476359982740992,
    "created": 1326476359982740992,
    "hash": "491734079555d64b6...58f63c5d7",
    "key": "/MyModel/FooBar",
    "parent": "000000000000000...000000000",
    "type": "MyModel"
}
```

bar

Hello World

after merge

```
>>> foo.merge(bar)
>>> foo.version.serialRepresentation.data()
{
    "attributes": {
        "first": {
            "updated": 1326476344447048960,
            "value": "Hello"
        },
        "second": {
            "updated": 1326476356582902016,
            "value": "World"
        }
    },
    "committed": 1326476824444653056,
    "created": 1326476347431011840,
    "hash": "06422c09770edf1b3c...bd8777544",
    "key": "/MyModel/FooBar",
    "parent": "8b52f06b5c2a6e5d...57da3a887",
    "type": "MyModel"
}
```

foo

Hello World

after merge

```
>>> foo.merge(bar)
>>> foo.version.serialRepresentation.data()
{
    "attributes": {
        "first": {
            "updated": 1326476344447048960,
            "value": "Hello"
        },
        "second": {
            "updated": 1326476356582902016,
            "value": "World"
        }
    },
    "committed": 1326476824444653056,
    "created": 1326476347431011840,
    "hash": "06422c09770edf1b3c...bd8777544",
    "key": "/MyModel/FooBar",
    "parent": "8b52f06b5c2a6e5d...57da3a887",
    "type": "MyModel"
}
```

foo

Attributes and MergeStrategies

```
>>> import dronestore
>>> from dronestore.merge import *
>>> from dronestore.attribute import *
>>>
>>> class Person(dronestore.Model):
...   first = StringAttribute(default="Firstname", strategy=LatestStrategy)
...   last = StringAttribute(default="Lastname", strategy=LatestStrategy)
...   phone = StringAttribute(default="N/A", strategy=LatestStrategy)
...   age = IntegerAttribute(default=0, strategy=MaxStrategy)
...   gender = StringAttribute(strategy=LatestObjectStrategy)
...
...   def __str__(self):
...     return '%s %s %s #%s age %d gender %s' % \
...       (self.key, self.first, self.last, self.phone, self.age, self.gender)
...
>>> p1 = Person('jbenet')
>>> p1.first = 'Juan'
>>> p1.gender = 'Female'
>>> p1.age = 23
>>> p1.commit()
>>>
>>> p2 = Person('jbenet')
>>> p2.last = 'Batiz-Benet'
>>> p2.phone = '1234567890'
>>> p2.gender = 'Male'
>>> p2.age = 18
>>> p2.commit()
>>>
>>> p1.merge(p2)
>>> print p1
/Person/jbenet Juan Batiz-Benet #1234567890 age 23 gender Male
```

Attributes and MergeStrategies

```
>>> import dronestore
>>> from dronestore.merge import *
>>> from dronestore.attribute import *
>>>
>>> class Person(dronestore.Model):
...   first = StringAttribute(default="Firstname", strategy=LatestStrategy)
...   last = StringAttribute(default="Lastname", strategy=LatestStrategy)
...   phone = StringAttribute(default="N/A", strategy=LatestStrategy)
...   age = IntegerAttribute(default=0, strategy=MaxStrategy)
...   gender = StringAttribute(strategy=LatestObjectStrategy)
...
...   def __str__(self):
...     return '%s %s %s #%s age %d gender %s' % \
...       (self.key, self.first, self.last, self.phone, self.age, self.gender)
...
>>> p1 = Person('jbenet')
>>> p1.first = 'Juan'
>>> p1.gender = 'Female'
>>> p1.age = 23
>>> p1.commit()
>>>
>>> p2 = Person('jbenet')
>>> p2.last = 'Batiz-Benet'
>>> p2.phone = '1234567890'
>>> p2.gender = 'Male'
>>> p2.age = 18
>>> p2.commit()
>>>
>>> p1.merge(p2)
>>> print p1
/Person/jbenet Juan Batiz-Benet #1234567890 age 23 gender Male
```

Attributes and MergeStrategies

```
>>> import dronestore
>>> from dronestore.merge import *
>>> from dronestore.attribute import *
>>>
>>> class Person(dronestore.Model):
...   first = StringAttribute(default="Firstname", strategy=LatestStrategy)
...   last = StringAttribute(default="Lastname", strategy=LatestStrategy)
...   phone = StringAttribute(default="N/A", strategy=LatestStrategy)
...   age = IntegerAttribute(default=0, strategy=MaxStrategy)
...   gender = StringAttribute(strategy=LatestObjectStrategy)
...
...   def __str__(self):
...     return '%s %s %s #%s age %d gender %s' % \
...       (self.key, self.first, self.last, self.phone, self.age, self.gender)
...
>>> p1 = Person('jbenet')
>>> p1.first = 'Juan'
>>> p1.gender = 'Female'
>>> p1.age = 23
>>> p1.commit()
>>>
>>> p2 = Person('jbenet')
>>> p2.last = 'Batiz-Benet'
>>> p2.phone = '1234567890'
>>> p2.gender = 'Male'
>>> p2.age = 18
>>> p2.commit()
>>>
>>> p1.merge(p2)
>>> print p1
/Person/jbenet Juan Batiz-Benet #1234567890 age 23 gender Male
```

Attributes and MergeStrategies

```
>>> import dronestore
>>> from dronestore.merge import *
>>> from dronestore.attribute import *
>>>
>>> class Person(dronestore.Model):
...   first = StringAttribute(default="Firstname", strategy=LatestStrategy)
...   last = StringAttribute(default="Lastname", strategy=LatestStrategy)
...   phone = StringAttribute(default="N/A", strategy=LatestStrategy)
...   age = IntegerAttribute(default=0, strategy=MaxStrategy)
...   gender = StringAttribute(strategy=LatestObjectStrategy)
...
...   def __str__(self):
...     return '%s %s %s #%s age %d gender %s' % \
...       (self.key, self.first, self.last, self.phone, self.age, self.gender)
...
>>> p1 = Person('jbenet')
>>> p1.first = 'Juan'
>>> p1.gender = 'Female'
>>> p1.age = 23
>>> p1.commit()
>>>
>>> p2 = Person('jbenet')
>>> p2.last = 'Batiz-Benet'
>>> p2.phone = '1234567890'
>>> p2.gender = 'Male'
>>> p2.age = 18
>>> p2.commit()
>>>
>>> p1.merge(p2)
>>> print p1
/Person/jbenet Juan Batiz-Benet #1234567890 age 23 gender Male
```

Attributes and MergeStrategies

```
>>> import dronestore
>>> from dronestore.merge import *
>>> from dronestore.attribute import *
>>>
>>> class Person(dronestore.Model):
...   first = StringAttribute(default="Firstname", strategy=LatestStrategy)
...   last = StringAttribute(default="Lastname", strategy=LatestStrategy)
...   phone = StringAttribute(default="N/A", strategy=LatestStrategy)
...   age = IntegerAttribute(default=0, strategy=MaxStrategy)
...   gender = StringAttribute(strategy=LatestObjectStrategy)
...
...   def __str__(self):
...     return '%s %s %s #%s age %d gender %s' % \
...       (self.key, self.first, self.last, self.phone, self.age, self.gender)
...
>>> p1 = Person('jbenet')
>>> p1.first = 'Juan'
>>> p1.gender = 'Female'
>>> p1.age = 23
>>> p1.commit()
>>>
>>> p2 = Person('jbenet')
>>> p2.last = 'Batiz-Benet'
>>> p2.phone = '1234567890'
>>> p2.gender = 'Male'
>>> p2.age = 18
>>> p2.commit()
>>>
>>> p1.merge(p2)
>>> print p1
/Person/jbenet Juan Batiz-Benet #1234567890 age 23 gender Male
```

Attributes and MergeStrategies

```
>>> p1.version.serialRepresentation.json()
{
    "attributes": {
        "age": {
            "value": 23
        },
        "first": {
            "updated": 1326477999232726016,
            "value": "Juan"
        },
        "gender": {
            "value": "Female"
        },
        "last": {
            "updated": 0,
            "value": "Lastname"
        },
        "phone": {
            "updated": 0,
            "value": "N/A"
        }
    },
    "committed": 1326477999233285888,
    "created": 1326477999233285888,
    "hash": "064c44ca45b614361...44e759076",
    "key": "/Person/jbenet",
    "parent": "000000000000000...000000000",
    "type": "Person"
}
```

p1

```
>>> p2.version.serialRepresentation.json()
{
    "attributes": {
        "age": {
            "value": 18
        },
        "first": {
            "updated": 0,
            "value": "Firstname"
        },
        "gender": {
            "value": "Male"
        },
        "last": {
            "updated": 1326477999233871104,
            "value": "Batiz-Benet"
        },
        "phone": {
            "updated": 1326477999234041856,
            "value": "1234567890"
        }
    },
    "committed": 1326477999234483968,
    "created": 1326477999234483968,
    "hash": "94c08a41d350884dd...fa1b2d02f",
    "key": "/Person/jbenet",
    "parent": "000000000000000...000000000",
    "type": "Person"
}
```

p2

Attributes and MergeStrategies

```
>>> p1.version.serialRepresentation.json()
{
    "attributes": {
        "age": {
            "value": 23
        },
        "first": {
            "updated": 1326477999232726016,
            "value": "Juan"
        },
        "gender": {
            "value": "Female"
        },
        "last": {
            "updated": 0,
            "value": "Lastname"
        },
        "phone": {
            "updated": 0,
            "value": "N/A"
        }
    },
    "committed": 1326477999233285888,
    "created": 1326477999233285888,
    "hash": "064c44ca45b614361...44e759076",
    "key": "/Person/jbenet",
    "parent": "000000000000000...000000000",
    "type": "Person"
}
```

p1

```
>>> p2.version.serialRepresentation.json()
{
    "attributes": {
        "age": {
            "value": 18
        },
        "first": {
            "updated": 0,
            "value": "Firstname"
        },
        "gender": {
            "value": "Male"
        },
        "last": {
            "updated": 1326477999233871104,
            "value": "Batiz-Benet"
        },
        "phone": {
            "updated": 1326477999234041856,
            "value": "1234567890"
        }
    },
    "committed": 1326477999234483968,
    "created": 1326477999234483968,
    "hash": "94c08a41d350884dd...fa1b2d02f",
    "key": "/Person/jbenet",
    "parent": "000000000000000...000000000",
    "type": "Person"
}
```

p2

Attributes and MergeStrategies

```
>>> p1.merge(p2)
>>> p1.version.serialRepresentation.json()
{
    "attributes": {
        "age": {
            "value": 23
        },
        "first": {
            "updated": 1326477999232726016,
            "value": "Juan"
        },
        "gender": {
            "value": "Male"
        },
        "last": {
            "updated": 1326477999233871104,
            "value": "Batiz-Benet"
        },
        "phone": {
            "updated": 1326477999234041856,
            "value": "1234567890"
        }
    },
    "committed": 1326478250984951040,
    "created": 1326477999233285888,
    "hash": "eaa32751cd59716a1...6941402ed",
    "key": "/Person/jbenet",
    "parent": "064c44ca45b6143...44e759076",
    "type": "Person"
}
```

p l

Repositories

```
>>> import redis
>>> import datastore
>>>
>>> class MyModel(dronestore.Model):
...   first = StringAttribute(strategy=LatestStrategy)
...   second = StringAttribute(strategy=LatestStrategy)
...
>>> from datastore.impl.redis import RedisDatastore
>>> ds = RedisDatastore(redis.Redis())
>>> drone = dronestore.Drone(ds)
>>>
>>> foo = MyModel('FooBar')
>>> foo.first = 'Hello'
>>> foo.commit()
>>>
>>> drone.merge(foo)
```

foo

```
>>> import redis
>>> import datastore
>>>
>>> class MyModel(dronestore.Model):
...   first = StringAttribute(strategy=LatestStrategy)
...   second = StringAttribute(strategy=LatestStrategy)
...
>>> from datastore.impl.redis import RedisDatastore
>>> ds = RedisDatastore(redis.Redis())
>>> drone = dronestore.Drone(ds)
>>>
>>> bar = MyModel('FooBar')
>>> bar.second = 'World'
>>> bar.commit()
>>>
>>> drone.merge(bar)
>>>
>>> foobar = drone.get(bar.key)
>>> print foobar.first, foobar.second
Hello World
```

bar

Repositories

```
>>> import redis
>>> import datastore
>>>
>>> class MyModel(dronestore.Model):
...   first = StringAttribute(strategy=LatestStrategy)
...   second = StringAttribute(strategy=LatestStrategy)
...
>>> from datastore.impl.redis import RedisDatastore
>>> ds = RedisDatastore(redis.Redis())
>>> drone = dronestore.Drone(ds)
>>>
>>> foo = MyModel('FooBar')
>>> foo.first = 'Hello'
>>> foo.commit()
>>>
>>> drone.merge(foo)
```

foo

```
>>> import redis
>>> import datastore
>>>
>>> class MyModel(dronestore.Model):
...   first = StringAttribute(strategy=LatestStrategy)
...   second = StringAttribute(strategy=LatestStrategy)
...
>>> from datastore.impl.redis import RedisDatastore
>>> ds = RedisDatastore(redis.Redis())
>>> drone = dronestore.Drone(ds)
>>>
>>> bar = MyModel('FooBar')
>>> bar.second = 'World'
>>> bar.commit()
>>>
>>> drone.merge(bar)
>>>
>>> foobar = drone.get(bar.key)
>>> print foobar.first, foobar.second
Hello World
```

bar

Repositories

```
>>> import redis
>>> import datastore
>>>
>>> class MyModel(dronestore.Model):
...   first = StringAttribute(strategy=LatestStrategy)
...   second = StringAttribute(strategy=LatestStrategy)
...
>>> from datastore.impl.redis import RedisDatastore
>>> ds = RedisDatastore(redis.Redis())
>>> drone = dronestore.Drone(ds)
>>>
>>> foo = MyModel('FooBar')
>>> foo.first = 'Hello'
>>> foo.commit()
>>>
>>> drone.merge(foo)
```

foo

```
>>> import redis
>>> import datastore
>>>
>>> class MyModel(dronestore.Model):
...   first = StringAttribute(strategy=LatestStrategy)
...   second = StringAttribute(strategy=LatestStrategy)
...
>>> from datastore.impl.redis import RedisDatastore
>>> ds = RedisDatastore(redis.Redis())
>>> drone = dronestore.Drone(ds)
>>>
>>> bar = MyModel('FooBar')
>>> bar.second = 'World'
>>> bar.commit()
>>>
>>> drone.merge(bar)
>>>
>>> foobar = drone.get(bar.key)
>>> print foobar.first, foobar.second
Hello World
```

bar

Repositories

```
>>> import redis
>>> import datastore
>>>
>>> class MyModel(dronestore.Model):
...   first = StringAttribute(strategy=LatestStrategy)
...   second = StringAttribute(strategy=LatestStrategy)
...
>>> from datastore.impl.redis import RedisDatastore
>>> ds = RedisDatastore(redis.Redis())
>>> drone = dronestore.Drone(ds)
>>>
>>> foo = MyModel('FooBar')
>>> foo.first = 'Hello'
>>> foo.commit()
>>>
>>> drone.merge(foo)
```

foo

```
>>> import redis
>>> import datastore
>>>
>>> class MyModel(dronestore.Model):
...   first = StringAttribute(strategy=LatestStrategy)
...   second = StringAttribute(strategy=LatestStrategy)
...
>>> from datastore.impl.redis import RedisDatastore
>>> ds = RedisDatastore(redis.Redis())
>>> drone = dronestore.Drone(ds)
>>>
>>> bar = MyModel('FooBar')
>>> bar.second = 'World'
>>> bar.commit()
>>>
>>> drone.merge(bar)
>>>
>>> foobar = drone.get(bar.key)
>>> print foobar.first, foobar.second
Hello World
```

bar

MergeStrategies

MergeStrategies

- Represent unique way to decide how the two values of a particular attribute merge together.

- Meant to enforce rules that help keep application semantics on attributes changed in multiple nodes.

- Can store state “in” the object (e.g. a timestamp).

```
class MaxStrategy(MergeStrategy):
  '''MaxStrategy picks the larger value.

  This Strategy stores no additional state.
  '''

  def merge(self, local_version, remote_version):

    attr_local = self._attribute_data(local_version)
    attr_remote = self._attribute_data(remote_version)

if not attr_remote:
  return None

if not attr_local:
  return attr_remote

if attr_remote['value'] > attr_local['value']:
  return attr_remote
return None # no change. keep local
```

```
class MaxStrategy(MergeStrategy):
  '''MaxStrategy picks the larger value.

  This Strategy stores no additional state.
  '''

  def merge(self, local_version, remote_version):

    attr_local = self._attribute_data(local_version)
    attr_remote = self._attribute_data(remote_version)

    if not attr_remote:
      return None

    if not attr_local:
      return attr_remote

    if attr_remote['value'] > attr_local['value']:
      return attr_remote
    return None # no change. keep local
```

```
class LatestStrategy(MergeStrategy):
  '''LatestStrategy merges attributes based solely on timestamp. In essence, the
  most recently written attribute wins.

  This Strategy stores its state thus:
  { 'updated' : nanotime.nanotime, 'value': attrValue }

  A value with a timestamp will be preferred over values without.
  '''

  REQUIRES_STATE = True

  def merge(self, local_version, remote_version):

    attr_local = self._attribute_data(local_version)
    attr_remote = self._attribute_data(remote_version)

    # if no timestamp found in remote. we're done!
    if not attr_remote or 'updated' not in attr_remote:
      return None

    # since other side has a timestamp, if we don't, take theirs.
    if not attr_local or 'updated' not in attr_local:
      return attr_remote

    # if we havent decided (both have timestamps), compare timestamps
    if attr_remote['updated'] > attr_local['updated']:
      return attr_remote
    return None # no change. keep local

def setAttribute(self, instance, rawData, default=False):
  '''Called whenever this particular attribute is set to a new value.'''
  # update the update metadata to reflect the current time.
  if default:
    rawData['updated'] = 0
  else:
    rawData['updated'] = nanotime.now().nanoseconds()
```

```
class LatestStrategy(MergeStrategy):
  '''LatestStrategy merges attributes based solely on timestamp. In essence, the
  most recently written attribute wins.

  This Strategy stores its state thus:
  { 'updated' : nanotime.nanotime, 'value': attrValue }

  A value with a timestamp will be preferred over values without.
  '''

  REQUIRES_STATE = True

  def merge(self, local_version, remote_version):

    attr_local = self._attribute_data(local_version)
    attr_remote = self._attribute_data(remote_version)

    # if no timestamp found in remote. we're done!
    if not attr_remote or 'updated' not in attr_remote:
      return None

    # since other side has a timestamp, if we don't, take theirs.
    if not attr_local or 'updated' not in attr_local:
      return attr_remote

    # if we havent decided (both have timestamps), compare timestamps
    if attr_remote['updated'] > attr_local['updated']:
      return attr_remote
    return None # no change. keep local

  def setAttribute(self, instance, rawData, default=False):
    '''Called whenever this particular attribute is set to a new value.'''
    # update the update metadata to reflect the current time.
    if default:
      rawData['updated'] = 0
    else:
      rawData['updated'] = nanotime.now().nanoseconds()
```

```
class LatestStrategy(MergeStrategy):
  '''LatestStrategy merges attributes based solely on timestamp. In essence, the
  most recently written attribute wins.

  This Strategy stores its state thus:
  { 'updated' : nanotime.nanotime, 'value': attrValue }

  A value with a timestamp will be preferred over values without.
  '''

  REQUIRES_STATE = True

  def merge(self, local_version, remote_version):

    attr_local = self._attribute_data(local_version)
    attr_remote = self._attribute_data(remote_version)

    # if no timestamp found in remote. we're done!
    if not attr_remote or 'updated' not in attr_remote:
      return None

    # since other side has a timestamp, if we don't, take theirs.
    if not attr_local or 'updated' not in attr_local:
      return attr_remote

    # if we havent decided (both have timestamps), compare timestamps
    if attr_remote['updated'] > attr_local['updated']:
      return attr_remote
    return None # no change. keep local

  def setAttribute(self, instance, rawData, default=False):
    '''Called whenever this particular attribute is set to a new value.'''
    # update the update metadata to reflect the current time.
    if default:
      rawData['updated'] = 0
    else:
      rawData['updated'] = nanotime.now().nanoseconds()
```

MergeStrategies

- LatestStrategy: last-written attribute value
- LatestObjectStrategy: last-written object
- MaxStrategy: larger attribute value
- MinStrategy: lesser attribute value

MergeStrategies

- are the crux of DroneStore and merging.
- spell out the semantics of the data model.
- “LatestStrategy” is not always best.
- Collections are tricky! (tombstones)

nanotime

ns precision unix timestamp

https://github.com/jbenet/nanotime

Nanotime provides a time object that keeps time as the number of nanoseconds since the UNIX epoch. In other words, it is a 64bit UNIX timestamp with nanosecond precision.

Why yet another time type!?

Over and over, I found the need for a portable, easy to process time type. The good old Unix timestamp was perfect except for the lack of precision. Finding no better alternative, I decided to make my own to fit my needs: a uint64 count of nanoseconds since the unix epoch.

What about `timespec` and `timeval`?

While `timevals` and `timespecs` add the much needed precision (`usec` and `nsec`), they do so in a cumbersome way. They store the extra precision in a second `uint32` rather than in one whole field. While this preserves quick access to the `sec` part of the time value, adding and subtracting `timespecs` and `timevals` is not as simple as integer operations. For most of my use cases, I found myself comparing, adding, subtracting time values much more than outputting them. Furthermore, I tend to require the sub-second precision when printing out, storing, and transferring (or anything else the `sec` field used to satisfy) anyway.

Range:

2 ^ 64 nanoseconds is roughly ~ 584.554531 years. The timespan between 1970 to 2554 is more than enough for me.

Then again, it may be best to use a range of 292 around 1970. If I find the need to go much earlier than 1970 in a non-textual representation, I may revise this.

Implementations

There are currently implementations in C, C++, Obj-C, Python.

License

Nanotime is under the MIT License.

datastore

unified API for multiple data stores

https://github.com/jbenet/datastore

datastore is a generic layer of abstraction for data store and database access. It is a **simple** API with the aim to enable application development in a datastore-agnostic way, allowing datastores to be swapped seamlessly without changing application code. Thus, one can leverage different datastores with different strengths without committing the application to one datastore throughout its lifetime. It looks like this:

```
+---------------+
|  application  |    <--- No cumbersome SQL or Mongo specific queries!
+---------------+
        |           <--- simple datastore API calls
+---------------+
|   datastore   |    <--- datastore implementation for underlying db
+---------------+
        |           <--- database specific calls
+---------------+
|  various dbs  |    <--- MySQL, Redis, MongoDB, FS, ...
+---------------+
```

In addition, grouped datastores significantly simplify interesting data access patterns (such as caching and sharding).

name poll:

- DroneStore
- RepoStore
- VersionStore

thanks! any questions?

http://github.com/jbenet/py-dronestore
http://github.com/jbenet/ios-dronestore
http://github.com/jbenet/datastore

Juan Batiz-Benet
jbenet@cs.stanford.edu