ObjectFabric

A new programming model for the cloud

Cyprien Noel

ObjectFabric

- Founded February 2011 in Mountain View

FOUNDER INSTITUTE

- Technology derives from financial software

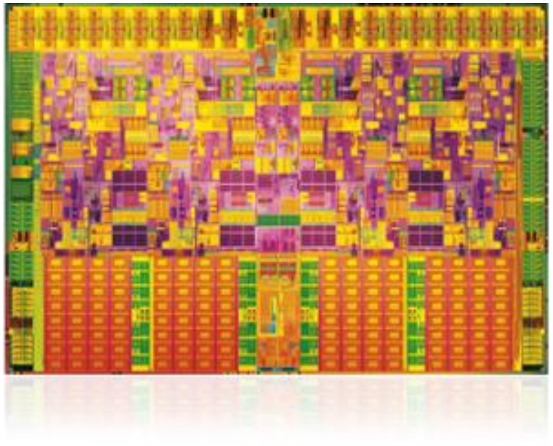
- 4-year development
- Open source, 3 contributors
- .NET apps, Web sites
- Previously xstm.org

Plan

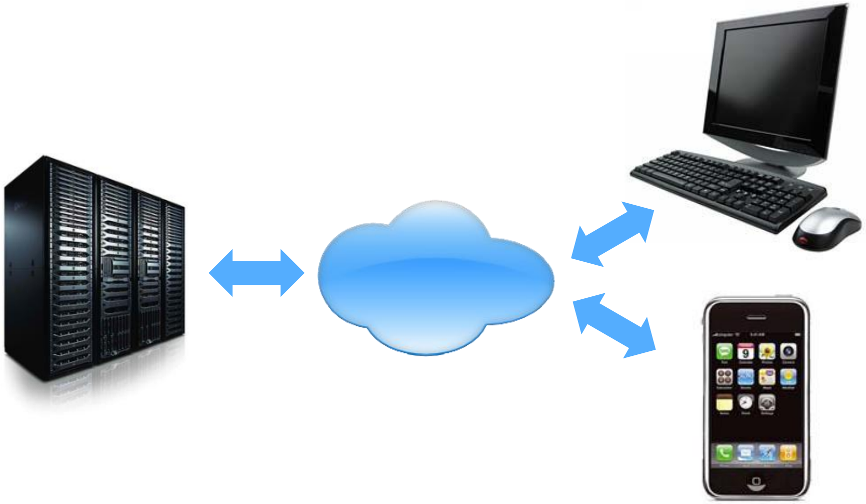
1. Challenges: Concurrency & Distribution
2. Transactional Memories
3. Distributed Transactional Memories
4. Our solution
   - Overview
   - Code samples
5. Thoughts (divagations!)
6. Questions

Challenges

Concurrency



Distribution



Models

Concurrency

- Locks
- "Lock Free"
  - CAS, Memory Barriers
- Message Passing
  - Message Loops, Actors
- Functional Programming
- Data Parallelism, GPUs
- Transactions
  - Transactional Memories

Distribution

- Message Passing
  - RPC, SOA
  - Queues, Actors
- Transactions
  - Databases

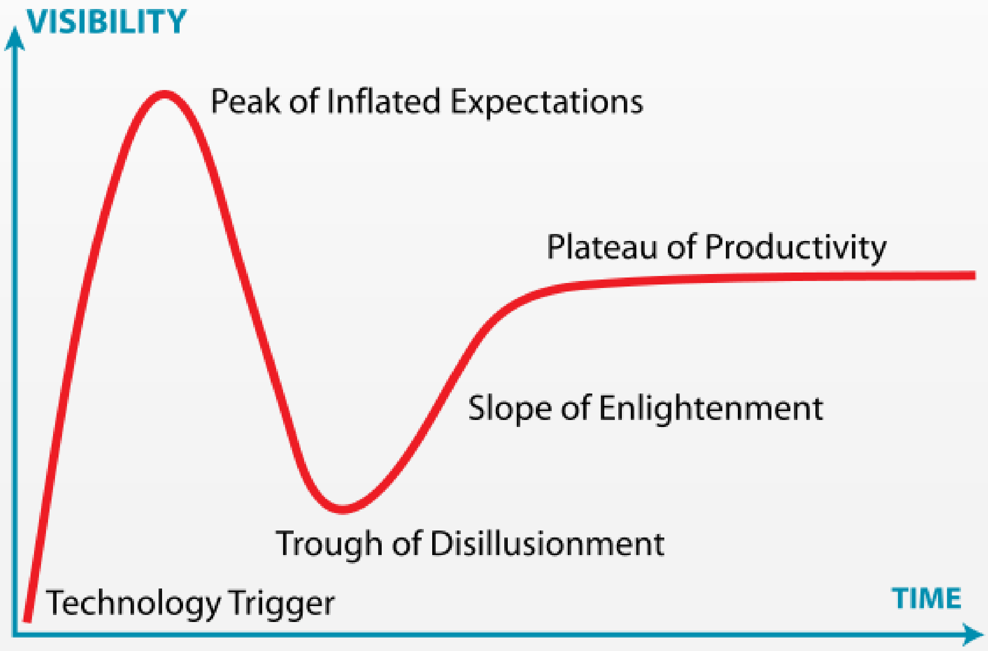
Transactional Memory

- Using transactions when modifying memory
  - Like a database but for in-memory objects
  - Sort of generalized CAS

Strengths	Weaknesses
Avoids locks pitfalls like deadlocks	Performance
Composable	Interactions with IO, locks
Rollback, no error recovery code	Debugging, language support

Transactional Memory

- More than 500 papers since first (Tim Harris, 2010)
  - *Transactional Memory: Architectural Support for Lock-Free Data Structures,* Maurice Herlihy, J. Eliot B. Moss, 1993



2004 to 2007 – Lots talks & impl.

2008 – First negative papers

2009 – Sun cancels Rock

2010 – Microsoft stops STM.net

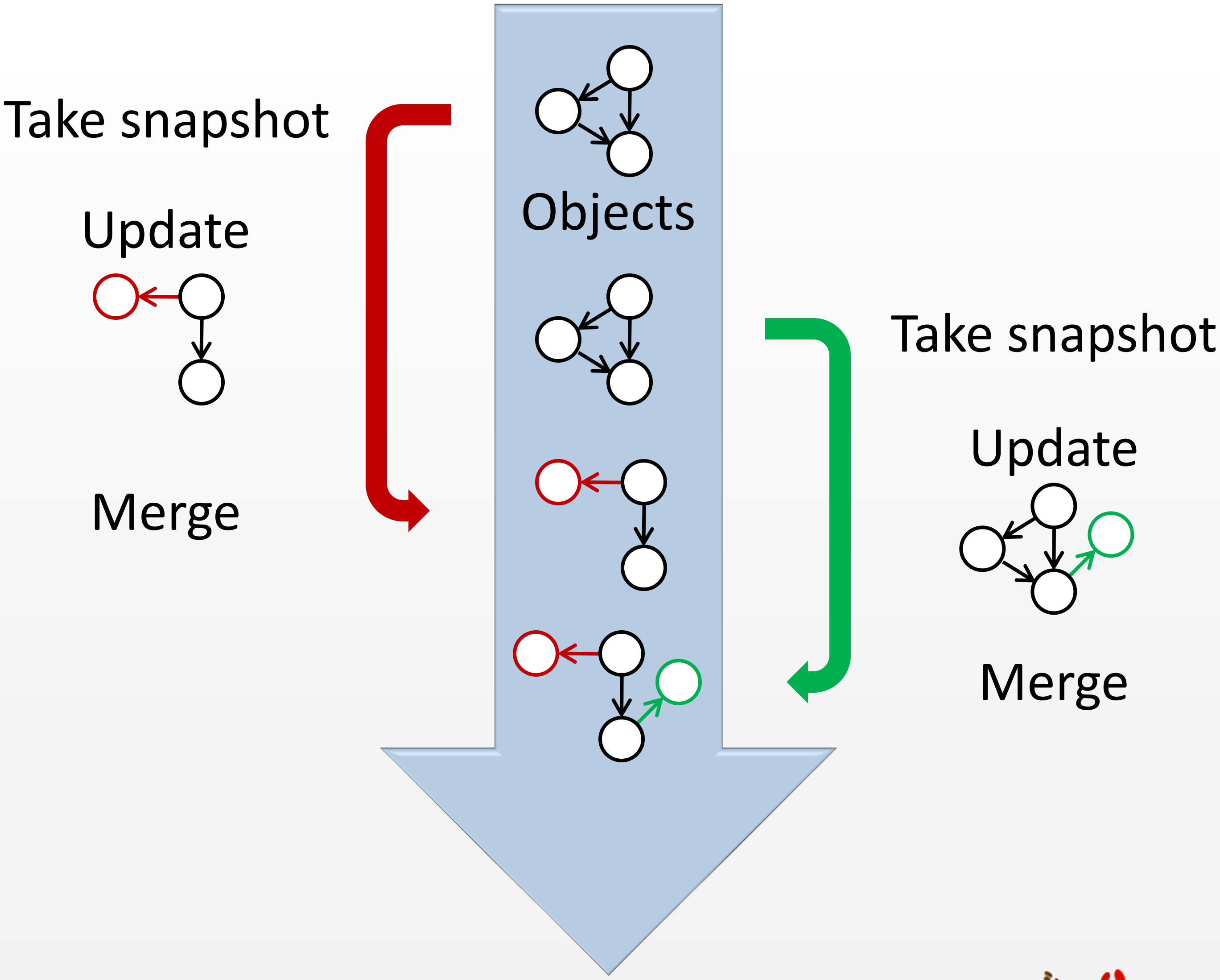
- Clojure, Multiverse, Scala, Fortress, TBoost.STM, CloudTM
- IBM BlueGene/Q processor

ObjectFabric

Transactional Memory

- Our implementation
  - For each tradeoff: pick easiest for developers
    - MVCC, View-Isolation & Opacity
    - No spurious aborts: can do IO
    - Strong isolation through type system
    - Lock free: guaranteed progress (starvation possible)
    - No byte code, JVM or lang. modif: easy debugging
- Modeled after source control systems
  - Intuitive to developers

Thread 1 Time Thread 2



Take snapshot

Update

Merge

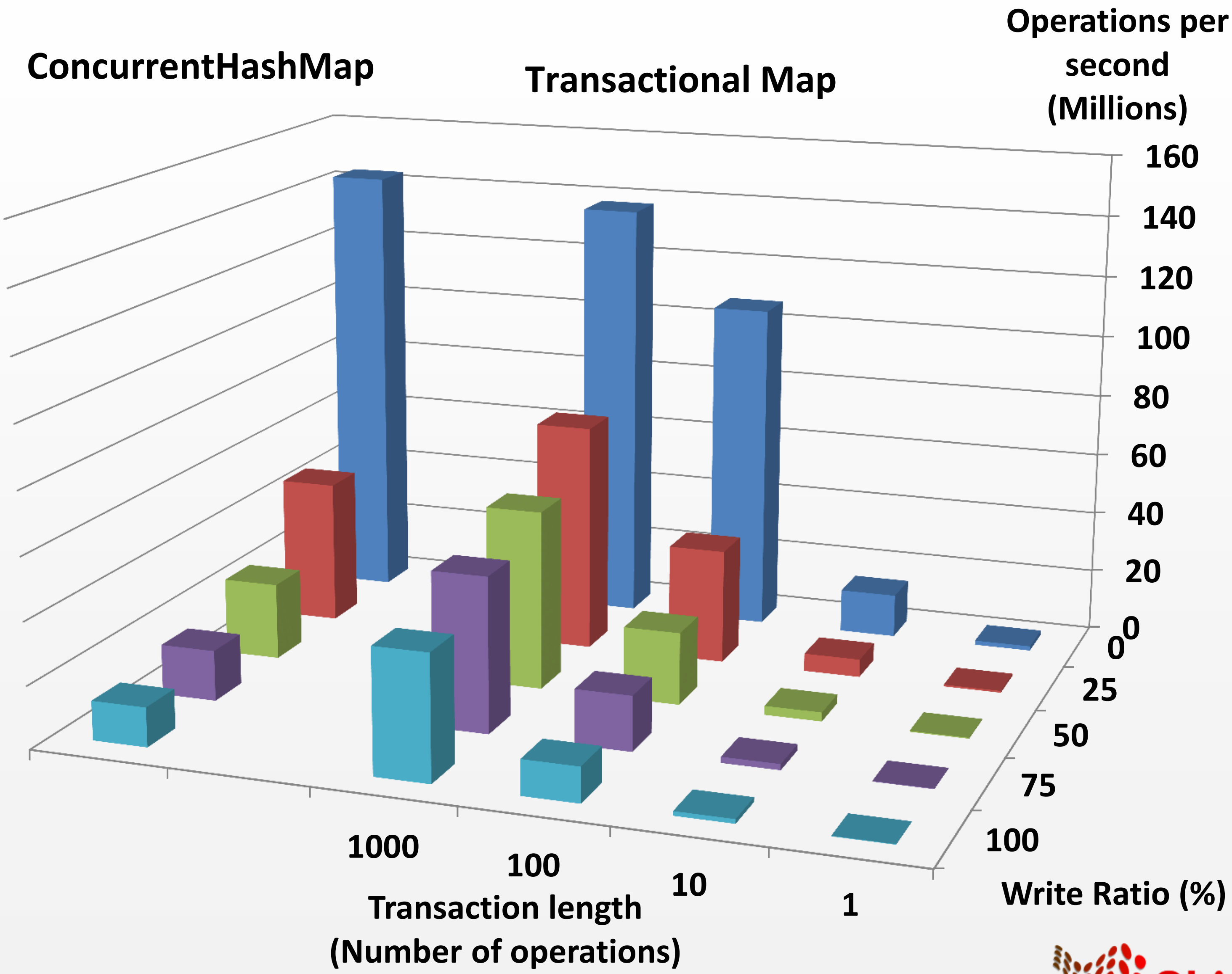
Objects

Take snapshot

Update

Merge

Performance



ObjectFabric

What else can be done with a TM?

- Transactions have read & write sets
  - Iterate write set -> e.g. logging, change notification

Thread A

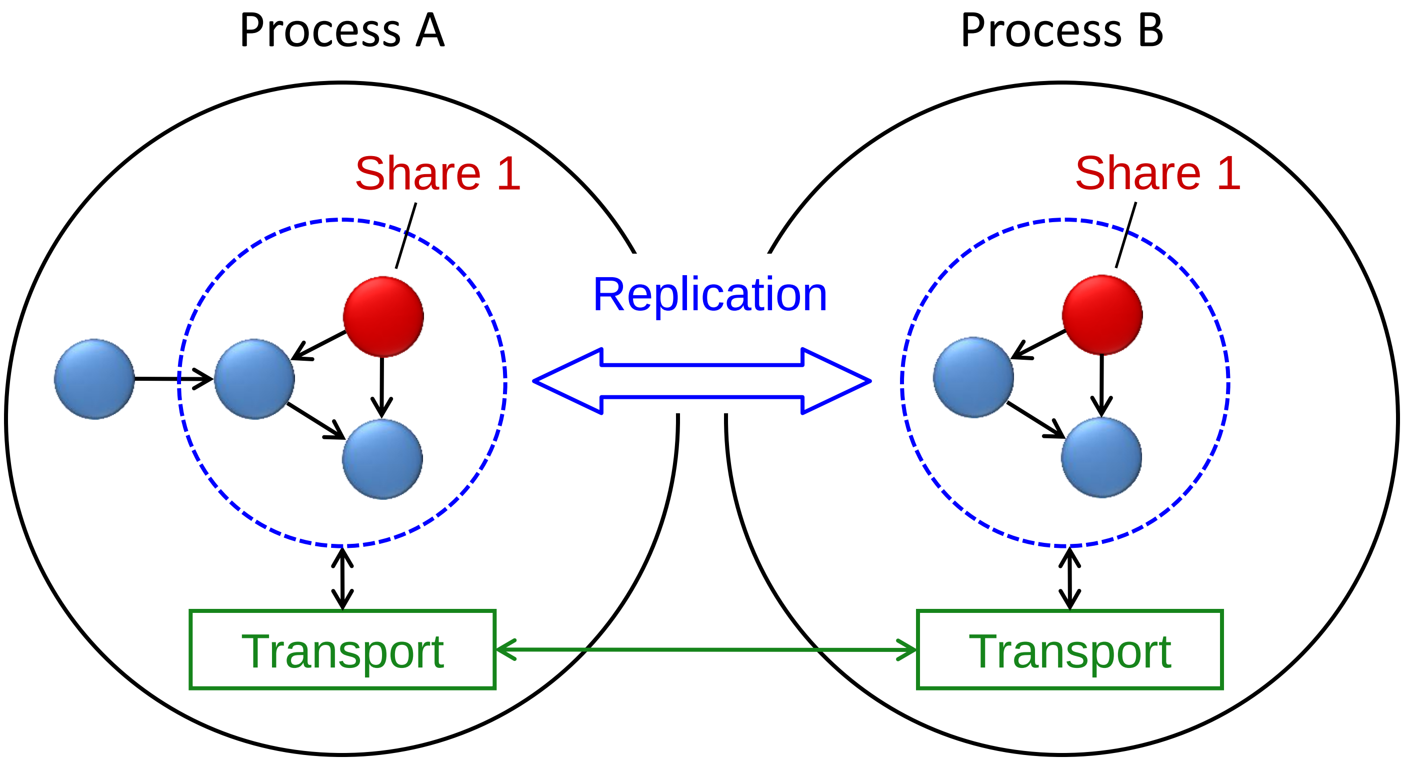
```
// Register a listener on the map
map.addListener(new KeyListener() {

   // A key has been put in the map
   public void onPut(Object key) {
      // Returns "value"
      map.get(key);
   }
}
```

Thread B

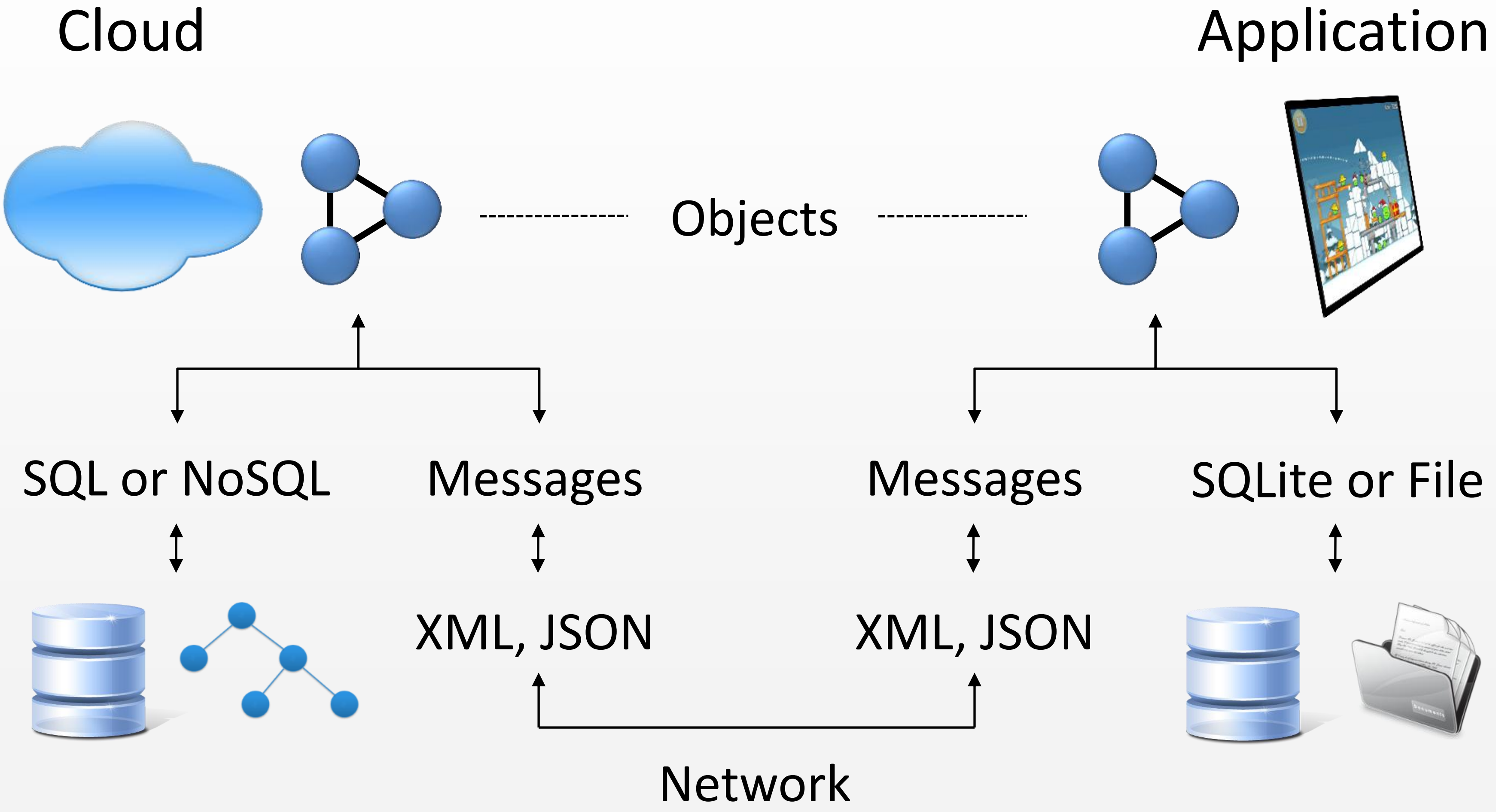
```
map.put("key", “value");
```

  - Persist writes to a store
  - Serialize and commit transactions remotely
- Extensions have small impact on performance
  - Extensions can skip to last update

Distributed Transactional Memory



Traditional Architecture



Cloud

Application

Objects

SQL or NoSQL

Messages

Messages

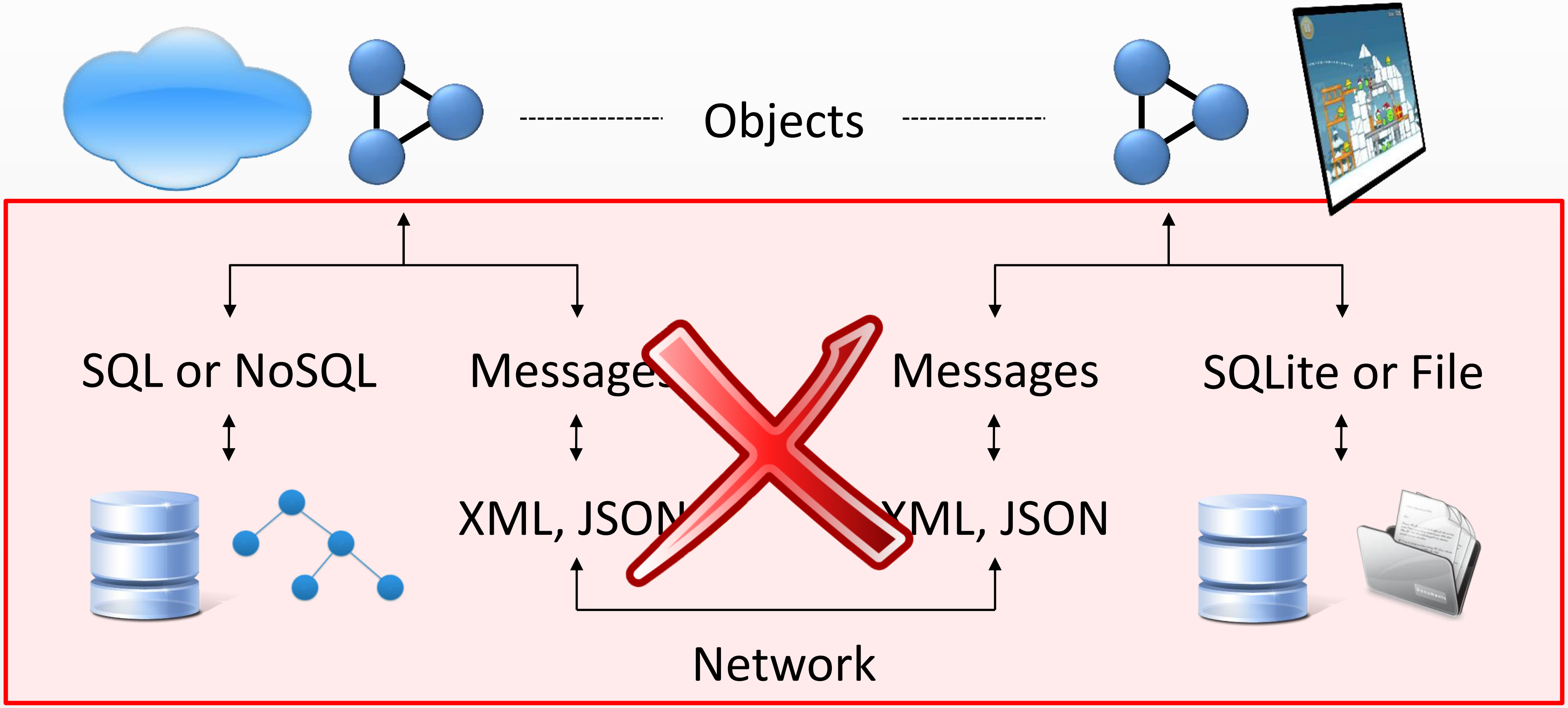
SQLite or File

XML, JSON

XML, JSON

Network

Traditional Architecture



Cloud

Application

Objects

SQL or NoSQL

Messages

Messages

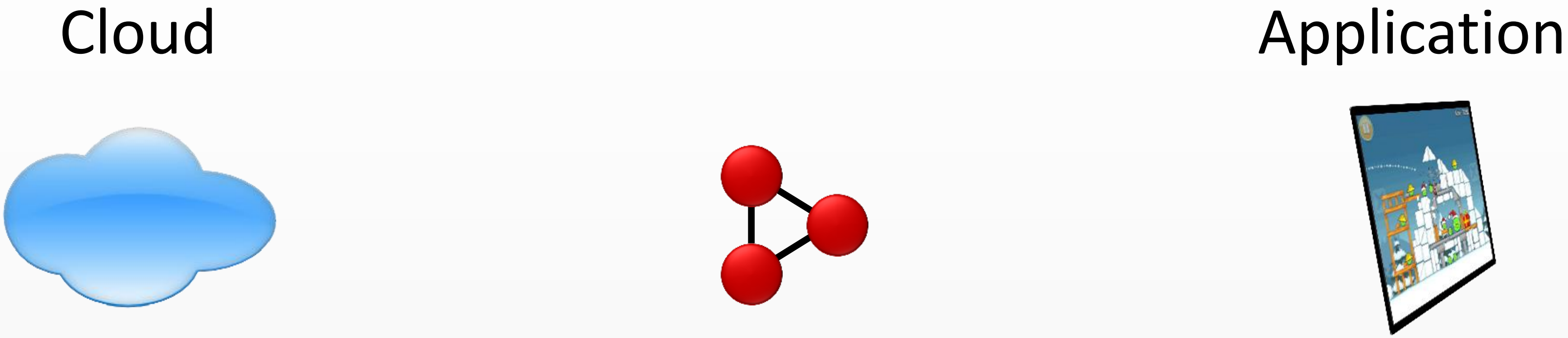
SQLite or File

XML, JSON

XML, JSON

Network

Distributed Transactional Memory



Developers only declare intents

- E.g. "Replicate this with server“
- E.g. "Make this durable"

Benefits

- A lot less or no code
- 10X faster than REST

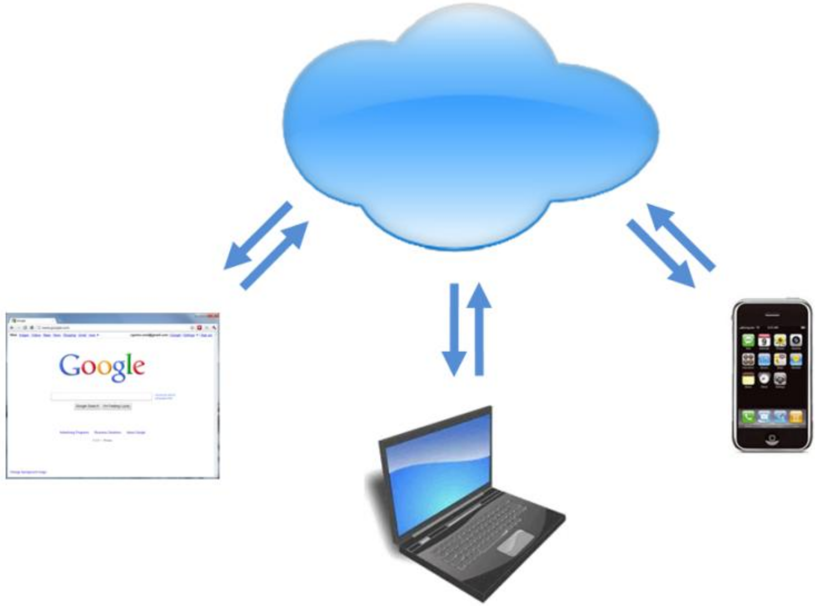
ObjectFabric

Distributed Transactional Memory

- Lots of interest over past 3 or 4 years
  - Many papers and implementations from academia
    - **Fénix, D^{2}STM** *(Technical University of Lisbon)*
    - **Arjuna** *(Newcastle University)*
    - **DMV** *(University of Toronto)*
    - **Isis^2** *(Cornell University)*
    - **DiSTM** *(University of Manchester)*
  - EU recently funded the **CloudTM** initiative in which **RedHat** is involved

Cloud·TM
A novel programming paradigm for the Cloud

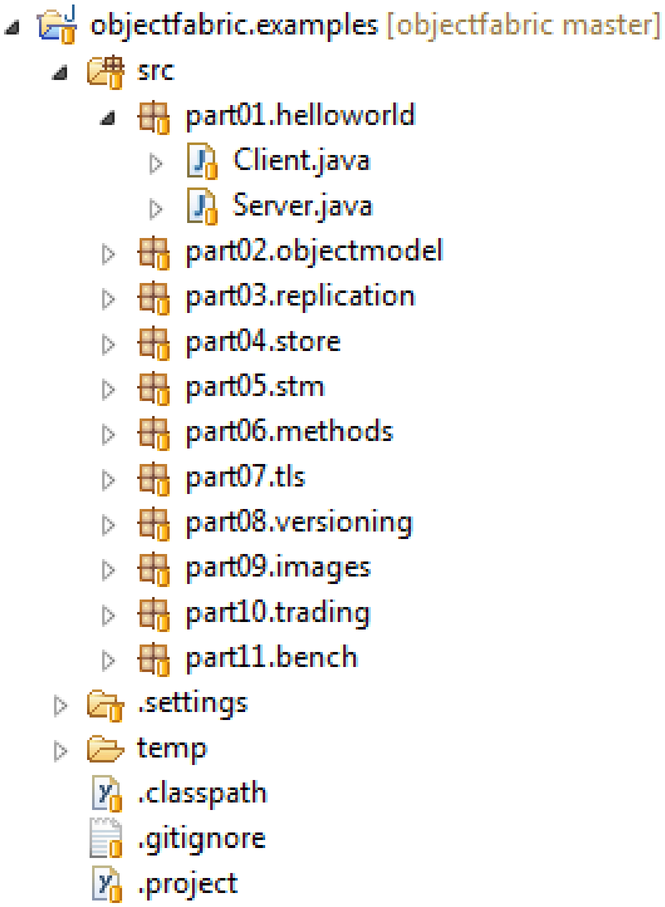
ObjectFabric



- Lightweight Java server
- Web, mobile & desktop
- Advanced stuff opt. in
- No configuration
- Java SE (Not EE, 1 jar)
- Secure (TLS, Shiro)
- Scalable (NIO)
- Fast (>100K message/s)

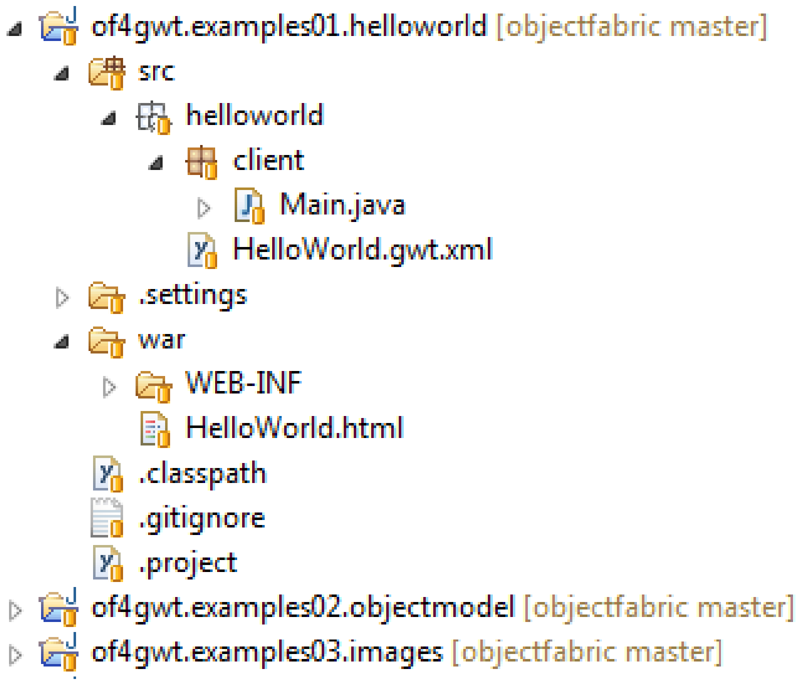
Get Started

Java



```
◢ objectfabric.examples [objectfabric master]
    ◢ src
        ◢ part01.helloworld
            ▷ Client.java
            ▷ Server.java
        ▷ part02.objectmodel
        ▷ part03.replication
        ▷ part04.store
        ▷ part05.stm
        ▷ part06.methods
        ▷ part07.tls
        ▷ part08.versioning
        ▷ part09.images
        ▷ part10.trading
        ▷ part11.bench
    ▷ .settings
    ▷ temp
      .classpath
      .gitignore
      .project
```

Google Web Toolkit



```
◢ of4gwt.examples01.helloworld [objectfabric master]
    ◢ src
        ◢ helloworld
            ◢ client
                ▷ Main.java
              HelloWorld.gwt.xml
    ▷ .settings
    ◢ war
        ▷ WEB-INF
          HelloWorld.html
      .classpath
      .gitignore
      .project
▷ of4gwt.examples02.objectmodel [objectfabric master]
▷ of4gwt.examples03.images [objectfabric master]
```

ObjectFabric

Server

```
// Open a port and listen for connections (Socket, or Comet)
SocketServer server = new SocketServer(8080);

server.setCallback(new Callback() {

    public void onConnection(SocketConnection session) {
        // Client connected, send an object
        session.send(new TMap ());
    }

    public void onDisconnection(SocketConnection session, Throwable t) {
        // Client disconnected
    }

    public void onReceived(SocketConnection session, Object object) {
        // Received an object from client
    }
}
```

```
client = new SocketClient("myCompany.com", 8080);

client.setCallback(new Callback() {

    public void onReceived(Object object) {
        // Received object from server
        TMap map = (TMap) object;
    }

    public void onDisconnected() {
        // Lost connection
    }
});

client.connectAsync(null);
```

Client

- Supported objects
  - Basic objects: "Immutable" (String, Integer, Date etc.)
  - Transactional collections: TMap, TList, TSet, LazyMap
  - Custom objects (Java or C# generated from description in XML or SQL)
- Once a collection or custom object is sent to another process, its state is replicated automatically
- Listeners work on replicas as in previous example

Process A (Client or server)

```
// Register a listener on the map
map.addListener(new KeyListener() {

   // A key has been put in the map
   public void onPut(Object key) {
      // Returns "value"
      map.get(key);
   }
}
```

Process B

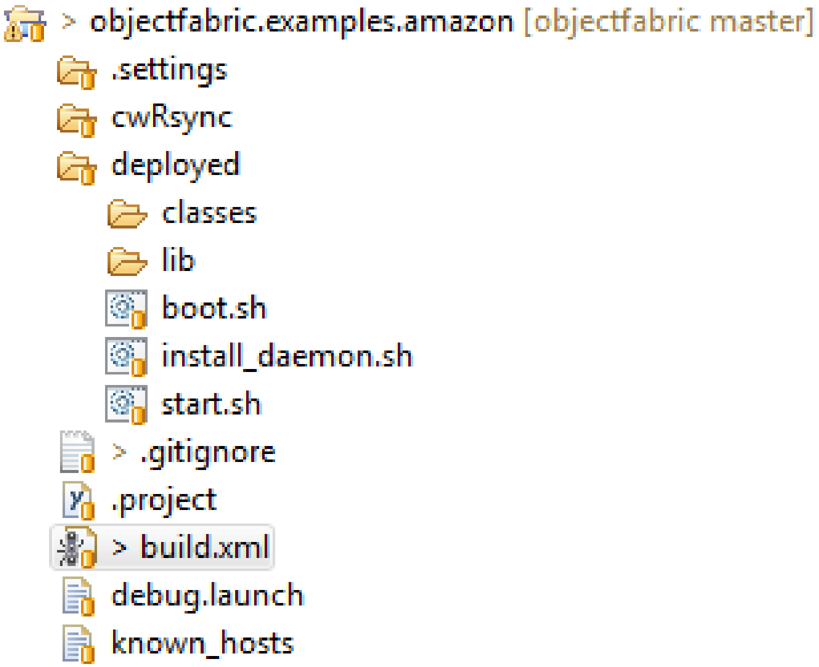
```
map.put("key", “value");
```

Transactional Memory

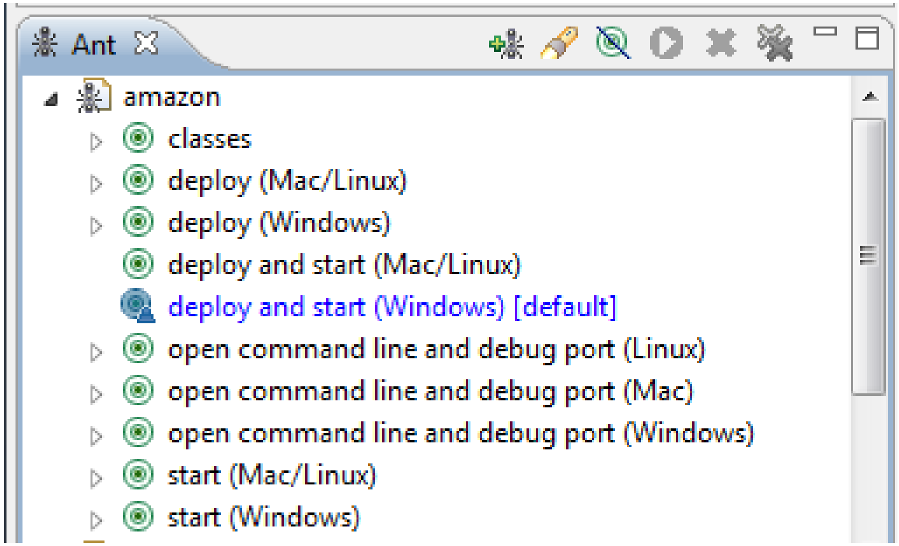
- Optional: transaction are started and committed automatically if needed when accessing collections or custom objects
- E.g. 2 objects **a** and **b**. This runs in parallel on several threads:

```
for (int i = 0; i < WRITE_COUNT; i++) {
    Transaction.run(new Runnable() {

        public void run() {
            assert a.getInt() == b.getInt();

            a.setInt(a.getInt() + 1);
            b.setInt(b.getInt() + 1);
        }
    });
}
```

Deploy to Amazon EC2



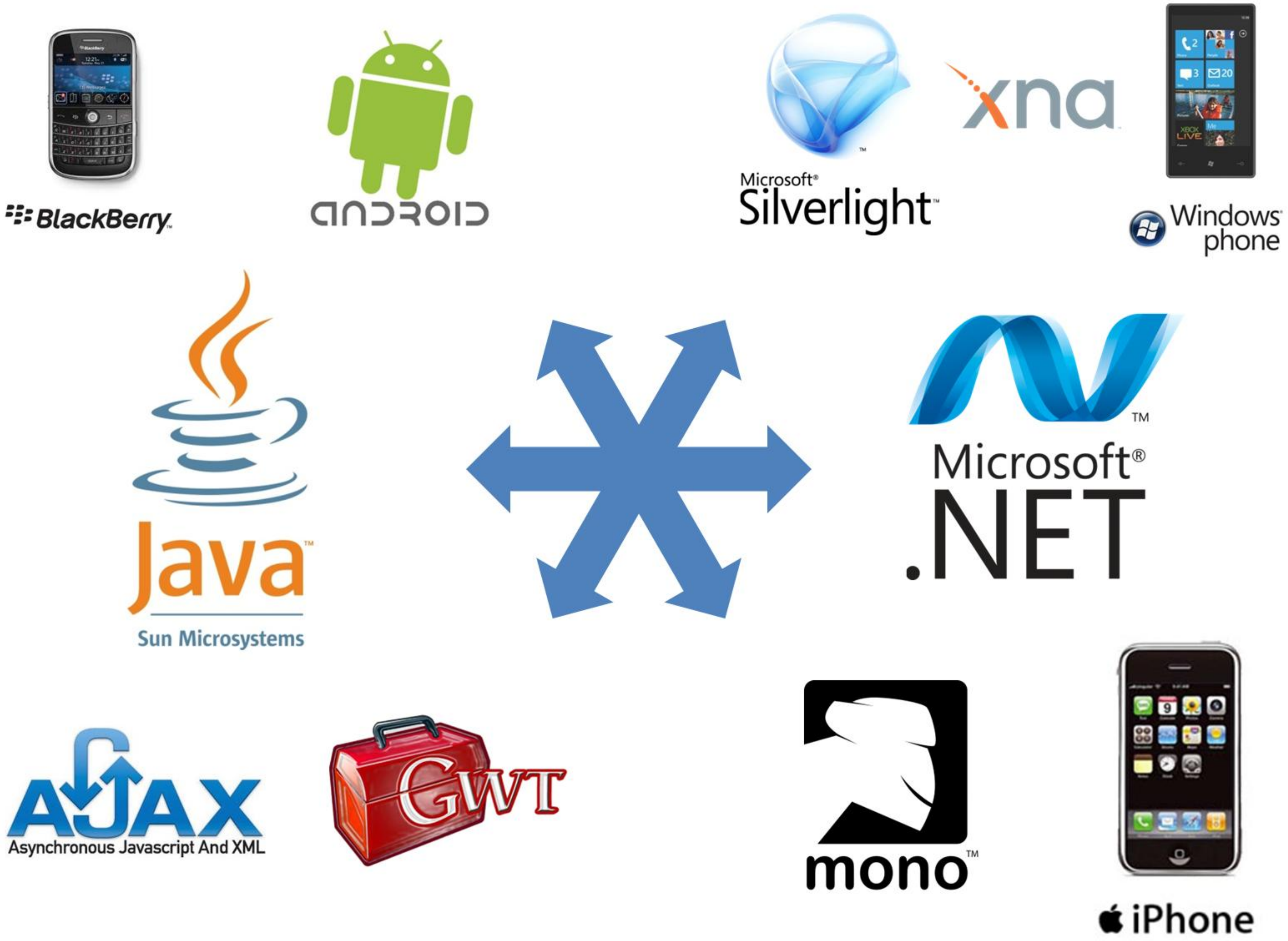
```
> objectfabric.examples.amazon [objectfabric master]
    .settings
    cwRsync
    deployed
        classes
        lib
        boot.sh
        install_daemon.sh
        start.sh
    > .gitignore
    .project
    > build.xml
    debug.launch
    known_hosts
```

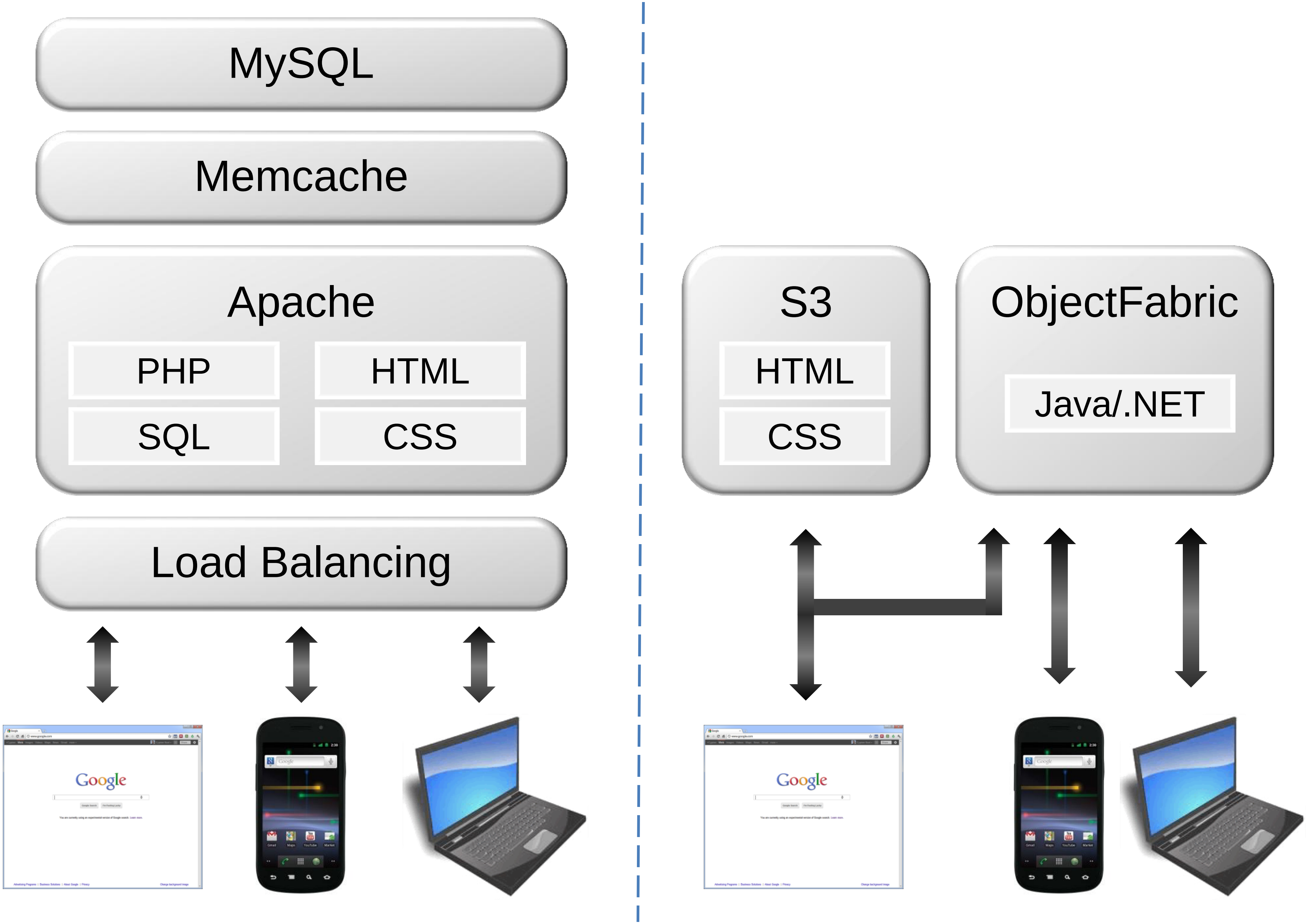


Ant

```
amazon
    classes
    deploy (Mac/Linux)
    deploy (Windows)
    deploy and start (Mac/Linux)
    deploy and start (Windows) [default]
    open command line and debug port (Linux)
    open command line and debug port (Mac)
    open command line and debug port (Windows)
    start (Mac/Linux)
    start (Windows)
```

Real-time interop. across entire cloud ecosystem





MySQL

Memcache

Apache

PHP

HTML

SQL

CSS

Load Balancing

S3

HTML

CSS

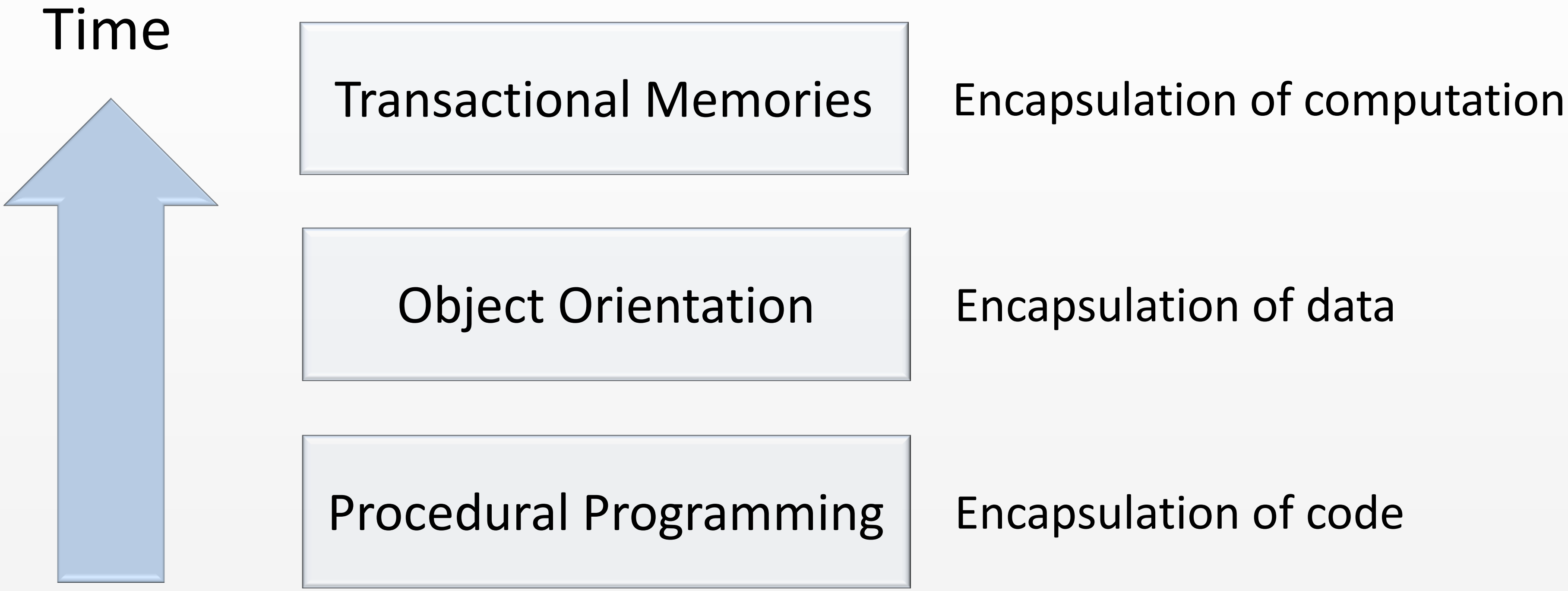
ObjectFabric

Java/.NET

Divagations!

- Transactions are the natural next step
- There are two rights ways to model the world
- Boundaries between objects and messages should not be imposed by technology

Transactions are the natural next step



Time

Transactional Memories	Encapsulation of computation
Object Orientation	Encapsulation of data
Procedural Programming	Encapsulation of code

Two right ways to model the world

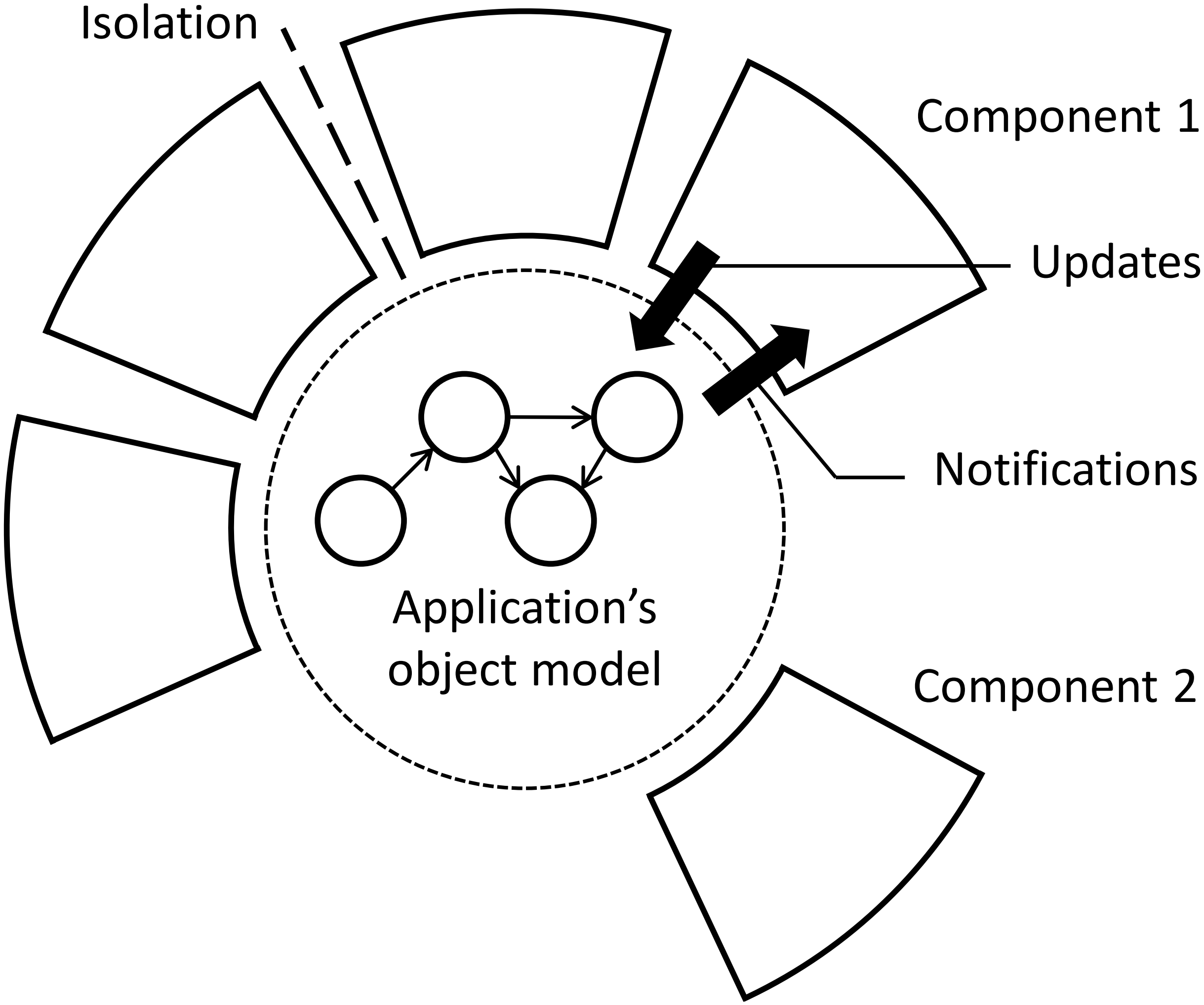
- Shared state mutated by transactions
  - Databases
  - Mainframes
  - Source controls
  - Transactional memories
    - Functional Programming
- Immutable messages between stateless processors
  - Actor Model
  - Message Bus, Queues etc.
  - Functional Programming

Two right ways to model the world

- "Objects" and "Messages"
  - Like frequency and time domains for sound
- The choice should be by domain or preference
  - Instead objects are confined to one process
  - Messages to "stitch" objects-based processes
  - In many applications domain modeled twice
- Previous example of UI connected to a server
  - Conversions Objects <-> Messages <-> Objects is wasteful and doesn't reduce coupling

Objects as a Bus

- Strengths of both shared state (OO, perf.) and Actors (safety)



Status / Road Map

- Java & GWT OK, .NET & Javascript in Beta
- Storage extensions
  - File based BTree (Ready)
  - SQL connector (Beta)
  - NoSQL connector (Alpha)
  - Offline Sync – HTML5/Mobile (Alpha)

- Scalability
  - Multi-Master, Eventual consistency
    - SimpleDB, ZooKeeper or Doozer
  - Goal: Hosted version (OaaS)

- Other
  - Specialized SDKs (Gaming, Chats)
  - REST connector (Alpha)

Thank You!

https://github.com/ObjectFabric

ObjectFabric