

APACHE GEODE

Introduction to Apache Geode *(incubating)*

APACHE:
BIG_DATA
EUROPE



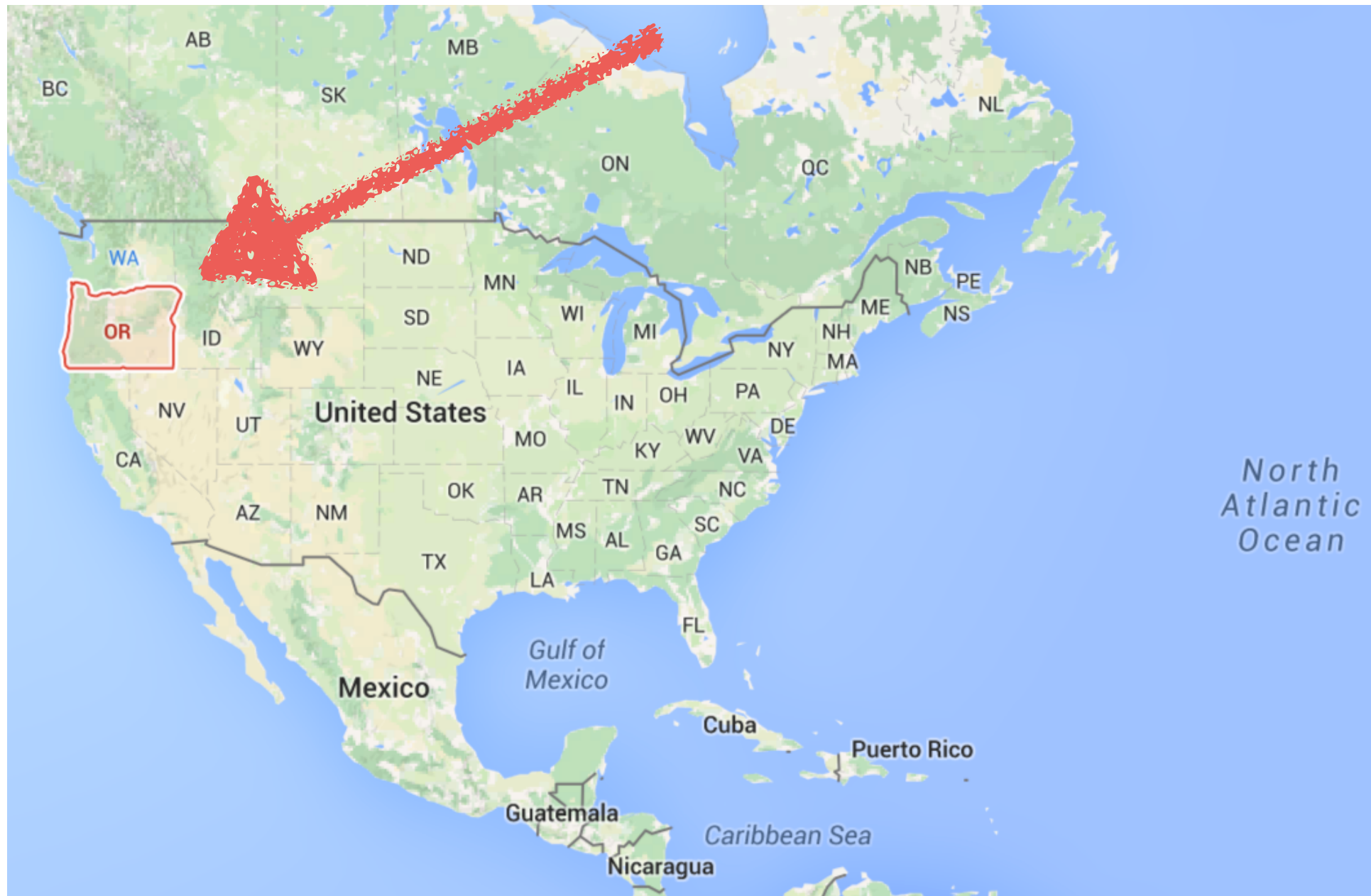
#ApacheConBigData

 The **Apache**
Software Foundation
Community-led development since 1999.

About us

Anthony Baker (@metatype)

William Markito (@william_markito)



Agenda

- Introduction to Geode
- Geode concepts
- The Geode open source source project
- Demo = Geode + Docker





Introduction

Apache Geode is...

*“...an in-memory, distributed database
with strong consistency
built to support low latency
transactional applications
at extreme scale.”*

- Massive increase in data volumes
- Falling margins per transaction
- Increasing cost of IT maintenance
- Need for elasticity in systems

2004

- Financial Services Providers (every major Wall Street bank)
- Department of Defense

- Real Time response needs
- Time to market constraints
- Need for flexible data models across enterprise
- Distributed development
- Persistence + In-memory

2008

- Largest travel Portal
- Airlines
- Trade clearing
- Online gambling

- Global data visibility needs
- Fast Ingest needs for data
- Need to allow devices to hook into enterprise data
- Always on

2014

- Largest Telcos
- Large mfrs
- Largest Payroll processor
- Auto insurance giants
- Largest rail systems on earth





China Railway Corporation

5,700 train stations

4.5 million tickets per day

20 million daily users

1.4 billion page views per day

40,000 visits per second



Indian Railways

7,000 stations

72,000 miles of track

23 million passengers daily

120,000 concurrent users

10,000 transactions per minute

*<http://pivotal.io/big-data/pivotal-gemfire>



China Railway
Corporation

Population: 1,401,586,609



Indian Railways

1,251,695,616

World: ~7,349,000,000

~36% of the world population

Application patterns

- Caching for speed and scale using read-through, write-through, and write-behind
- OLTP system of record with in-memory for speed, on disk for durability
- Parallel compute grid
- Real-time analytics

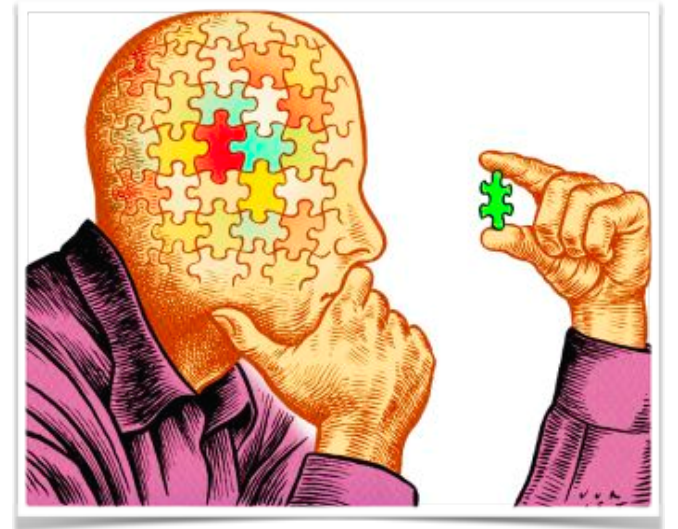


Concepts



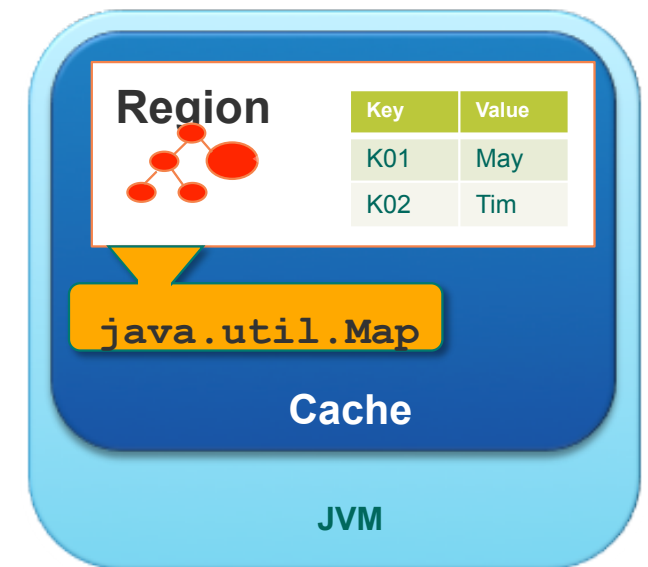
Geode concepts and usage

- Cache
- Region
- Member
- Client Cache
- Functions
- Listeners



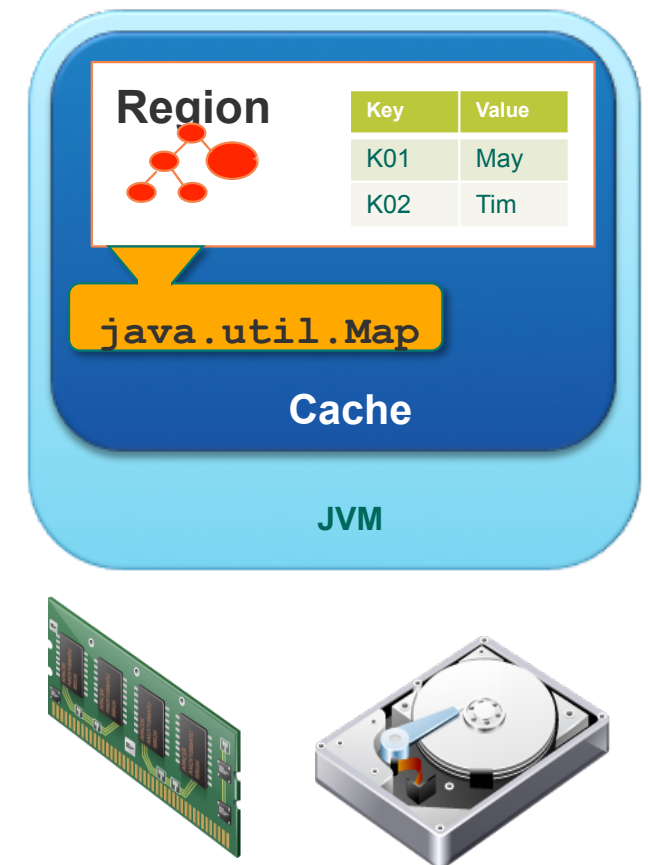
Concepts

- Cache
 - In-memory storage and management for your data
 - Configurable through XML, Spring, Java API, or CLI
 - Collection of **Region**



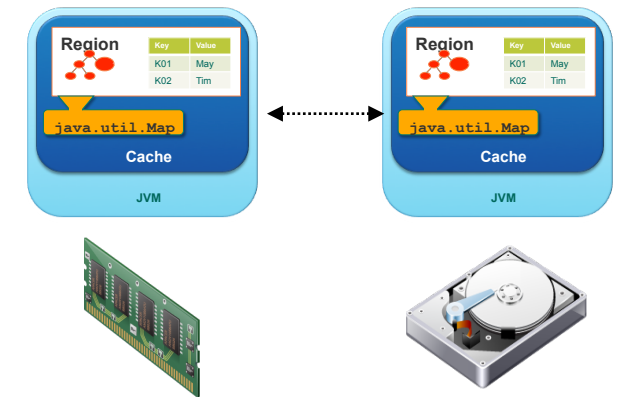
Concepts

- Region
 - Distributed `java.util.Map` on steroids (key/value)
 - Consistent API regardless of where or how data is stored
 - Observable (reactive)
 - Highly available, redundant on cache **Member(s)**
 - Querying



Concepts

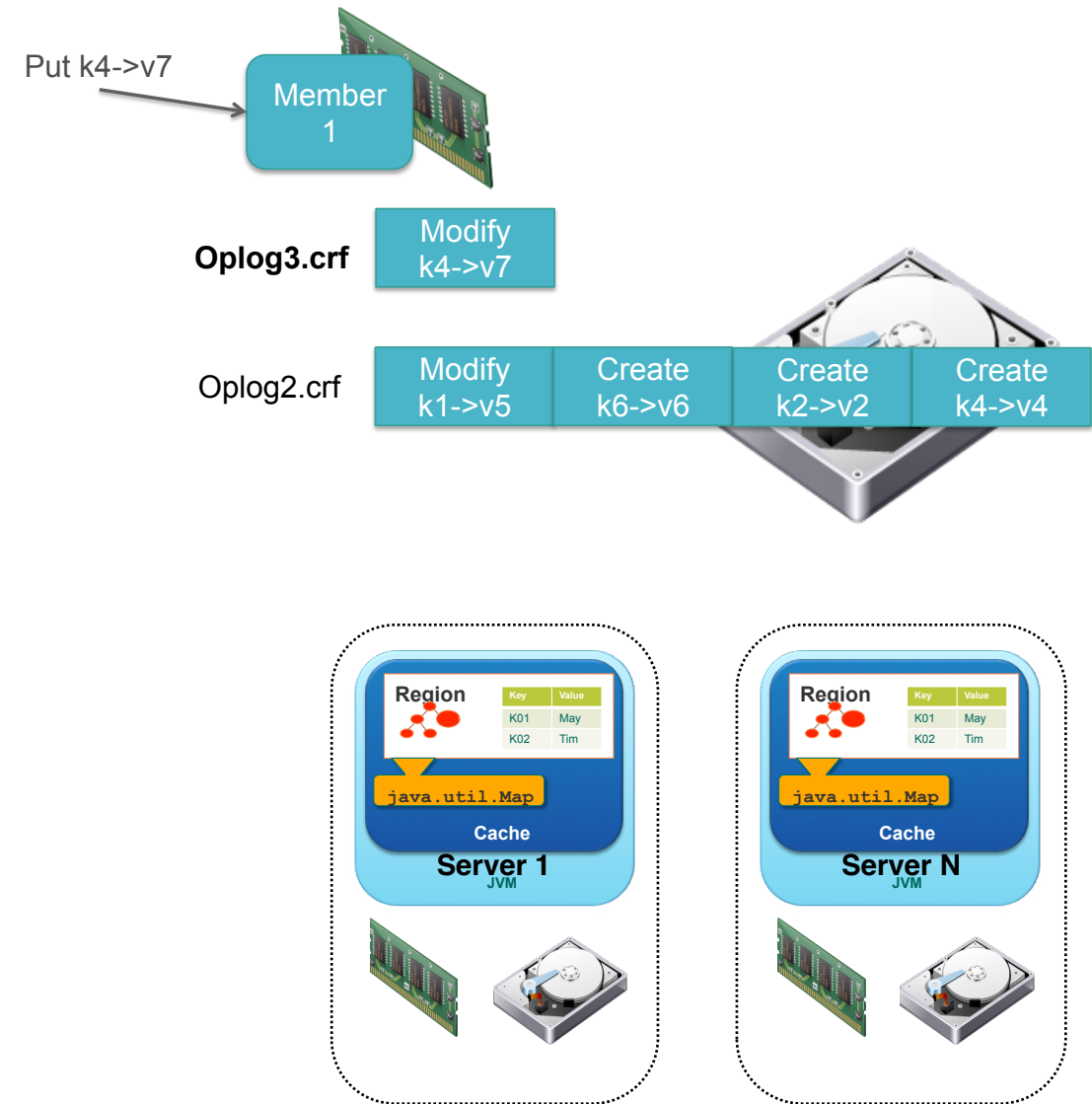
- Region
 - Local, replicated, or partitioned
 - In-memory or persistent
 - Redundant
 - LRU
 - Overflow



LOCAL
LOCAL_HEAP_LRU
LOCAL_OVERFLOW
LOCAL_PERSISTENT
LOCAL_PERSISTENT_OVERFLOW
PARTITION
PARTITION_HEAP_LRU
PARTITION_OVERFLOW
PARTITION_PERSISTENT
PARTITION_PERSISTENT_OVERFLOW
PARTITION_PROXY
PARTITION_PROXY_REDUNDANT
PARTITION_REDUNDANT
PARTITION_REDUNDANT_HEAP_LRU
PARTITION_REDUNDANT_OVERFLOW
PARTITION_REDUNDANT_PERSISTENT
PARTITION_REDUNDANT_PERSISTENT_OVERFLOW
REPLICATE
REPLICATE_HEAP_LRU
REPLICATE_OVERFLOW
REPLICATE_PERSISTENT
REPLICATE_PERSISTENT_OVERFLOW
REPLICATE_PROXY

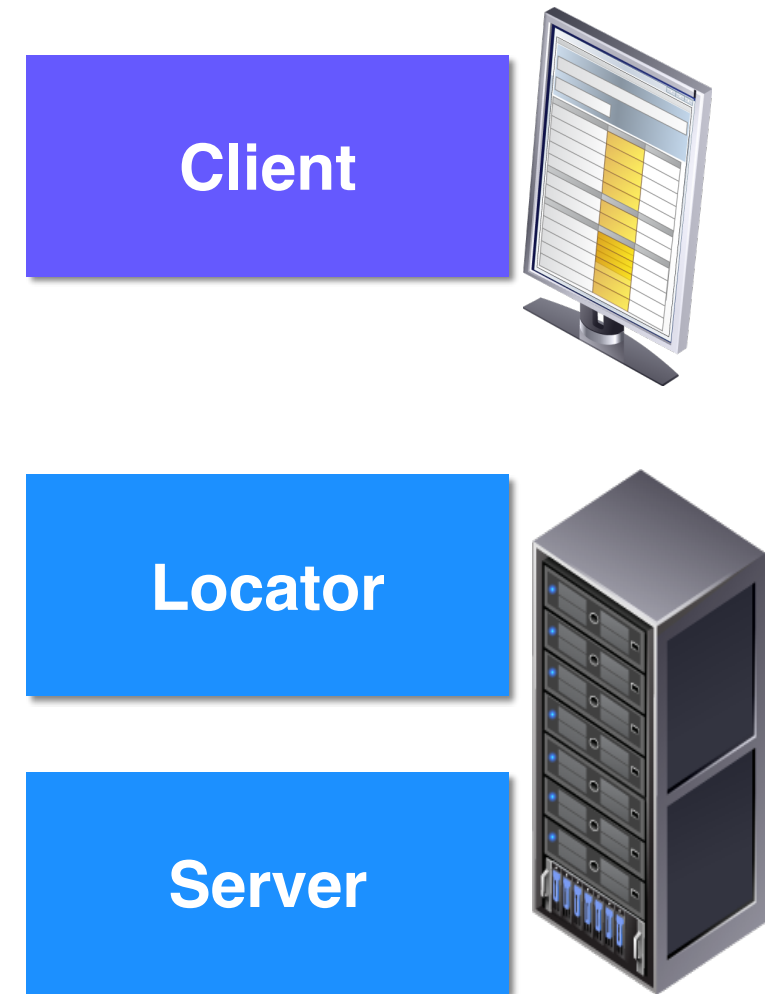
Concepts

- Persistent region
- Durability
- WAL for efficient writes
- Consistent recovery
- Compaction



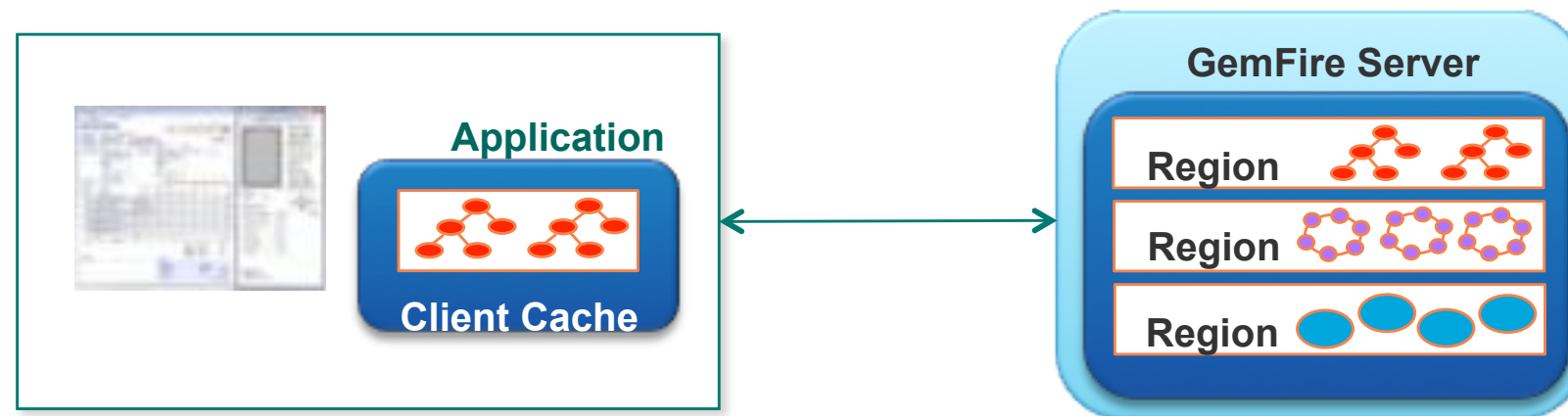
Concepts

- Member
 - A process that has a connection to the cluster
 - A process that has created a cache
 - Embeddable within your application



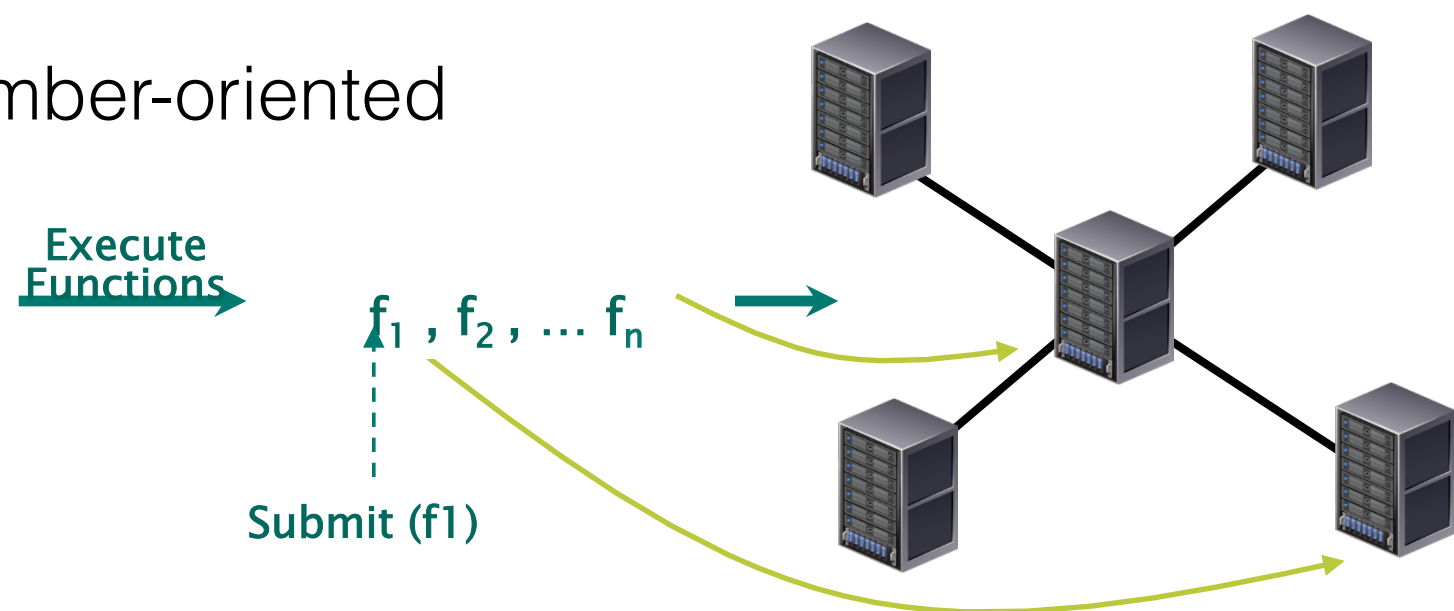
Concepts

- Client cache
 - A process connected to the Geode server(s)
 - Can have a local copy of the data
 - Can be notified about events in the cluster



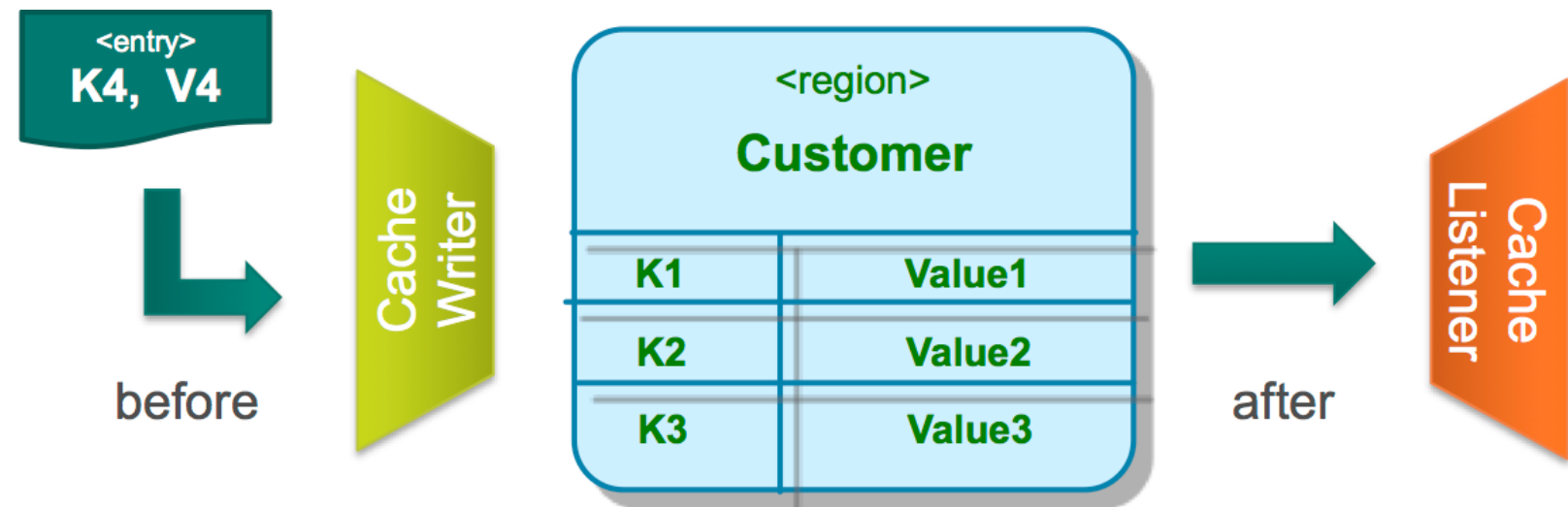
Concepts

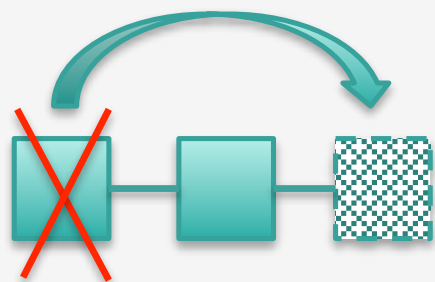
- Functions
 - Used for distributed concurrent processing (Map/Reduce, stored procedure, data parallel, ...)
 - Highly available
 - Data-oriented, member-oriented



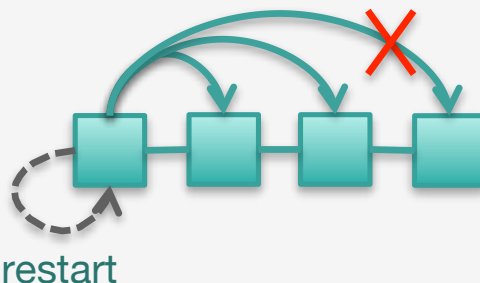
Concepts

- Listeners
 - CacheWriter / CacheListener
 - AsyncEventListener (queue / batch)
 - Parallel or Serial
 - Conflation

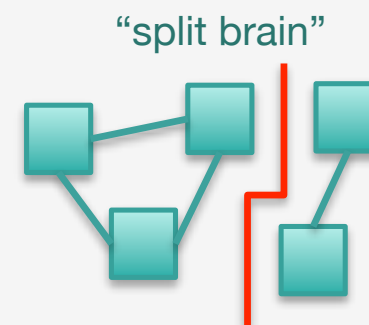




Failing data copies are replaced transparently



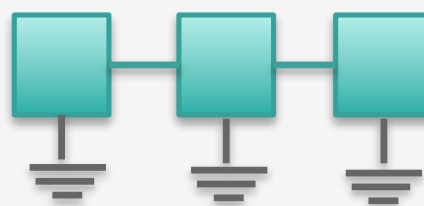
Failed function executions are restarted automatically



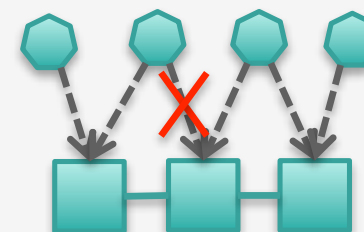
Network segmentations are identified and fixed automatically



Data is replicated to other clusters and sites (WAN)



Data is persisted on local disk for ultimate durability



Client and cluster disconnections are handled gracefully

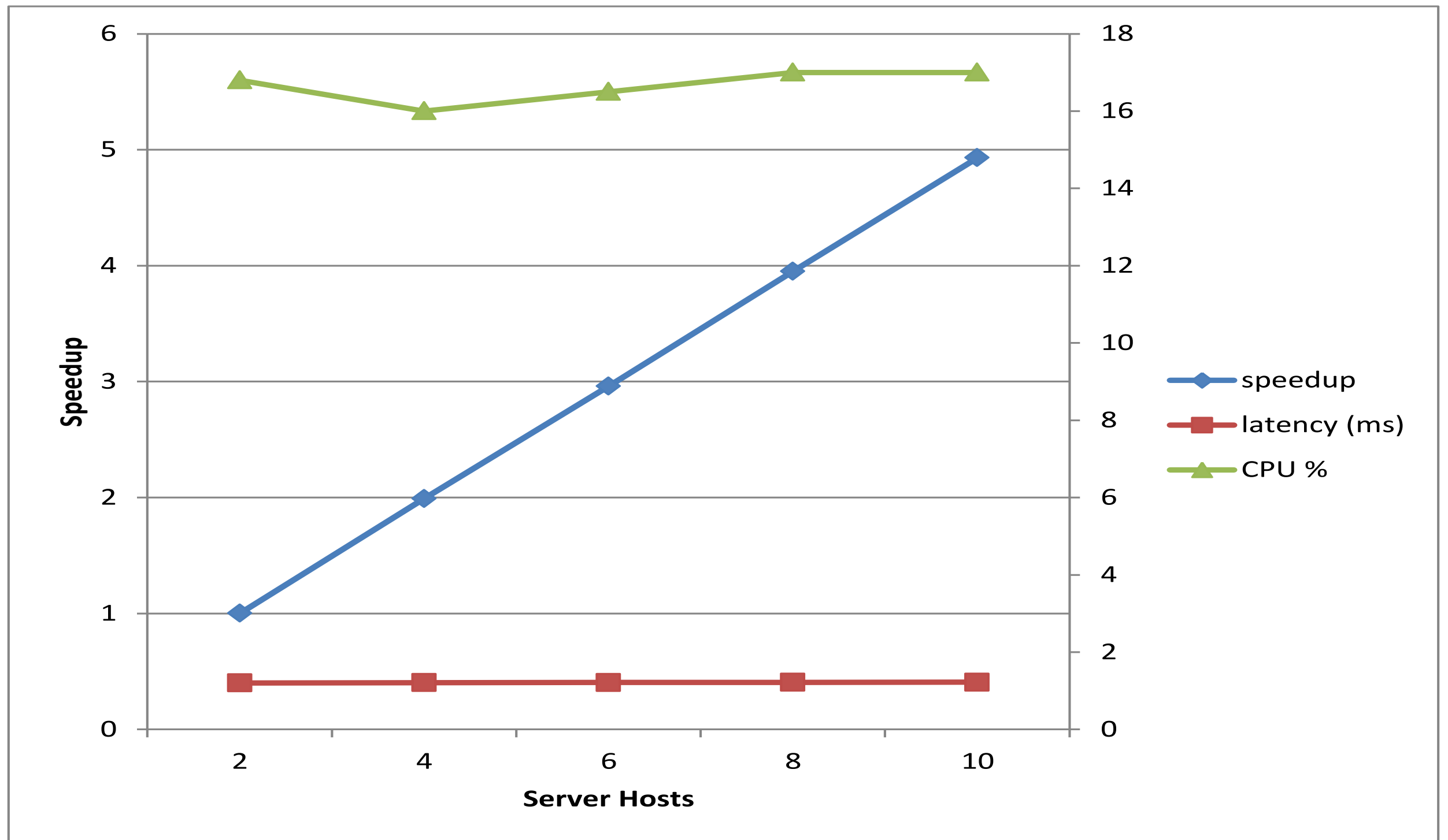
Pivotal

What makes it go fast?

- Minimize copying
- Minimize contention points
- Flexible consistency model
- Partitioning and parallelism
- Avoid disk seeks
- Automated benchmarks



Horizontal scaling for reads, consistent latency and CPU



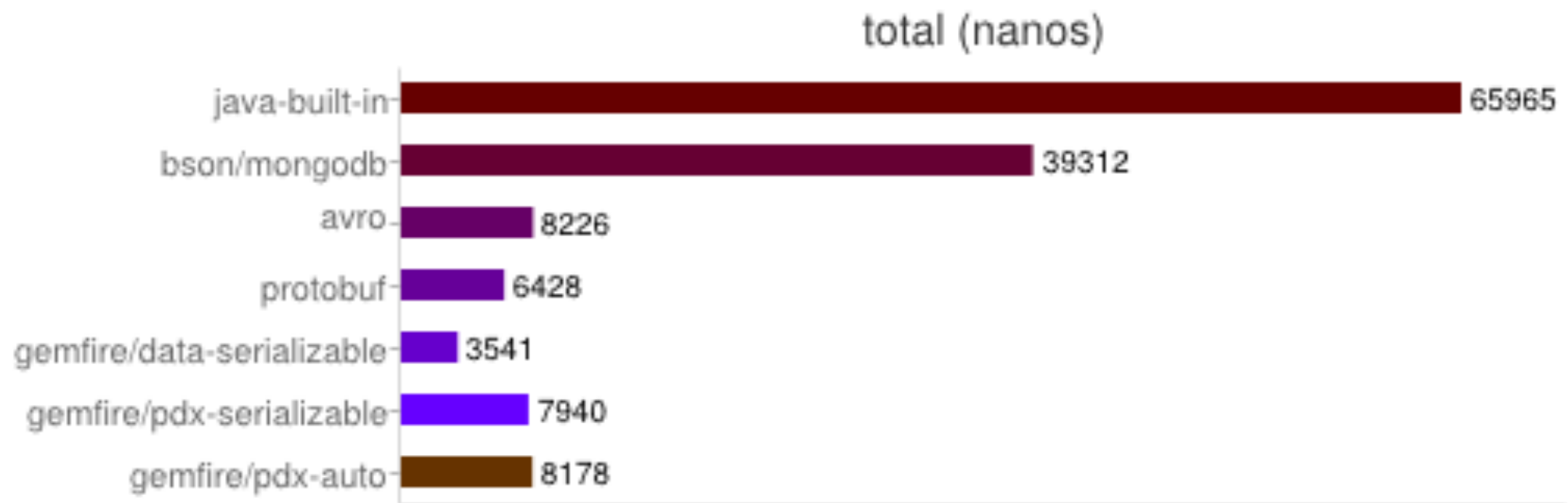
Fixed or flexible schema?

id	name	age	pet_id

or

```
{  
  id    : 1,  
  name  : "Fred",  
  age   : 42,  
  pet   : {  
    name : "Barney",  
    type : "dino"  
  }  
}
```

But how to serialize data?



C#, C++, Java, JSON

Portable Data eXchange

		header				data		
	pdx		length		dsid		typeid	
							fields	
							offsets	

No IDL, no schemas, no hand-coding

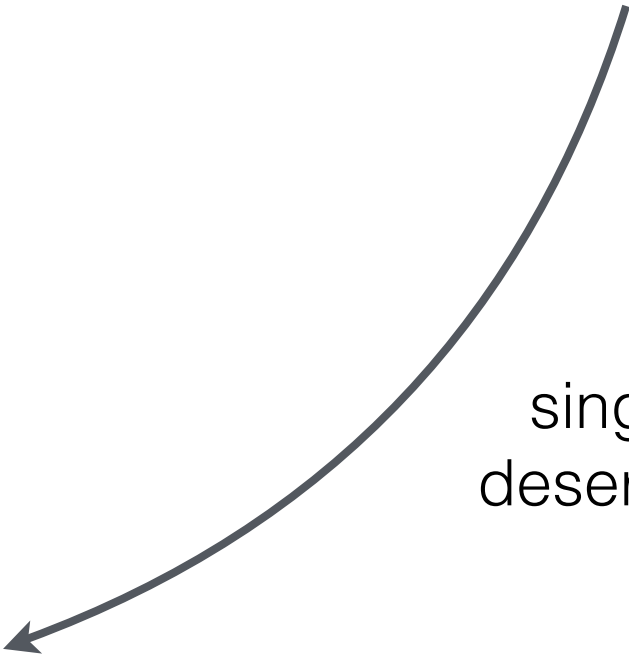
* domain object classes not required

Efficient for queries

```
SELECT p.name FROM /Person p WHERE p.pet.type = "dino"
```

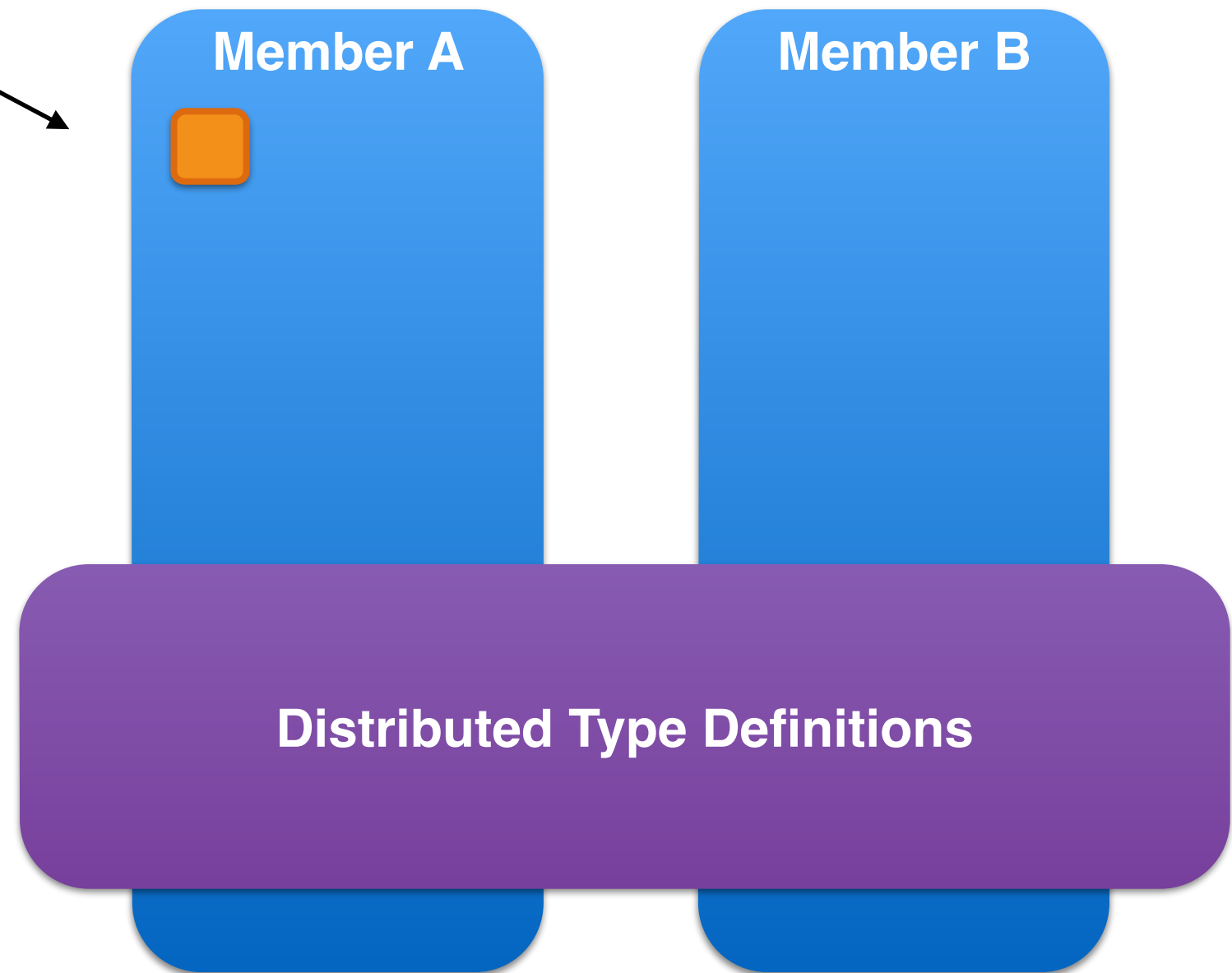
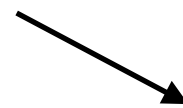
```
{  
  id    : 1,  
  name  : "Fred",  
  age   : 42,  
  pet   : {  
    name : "Barney",  
    type : "dino"  
  }  
}
```

single field
deserialization



Distributed type registry

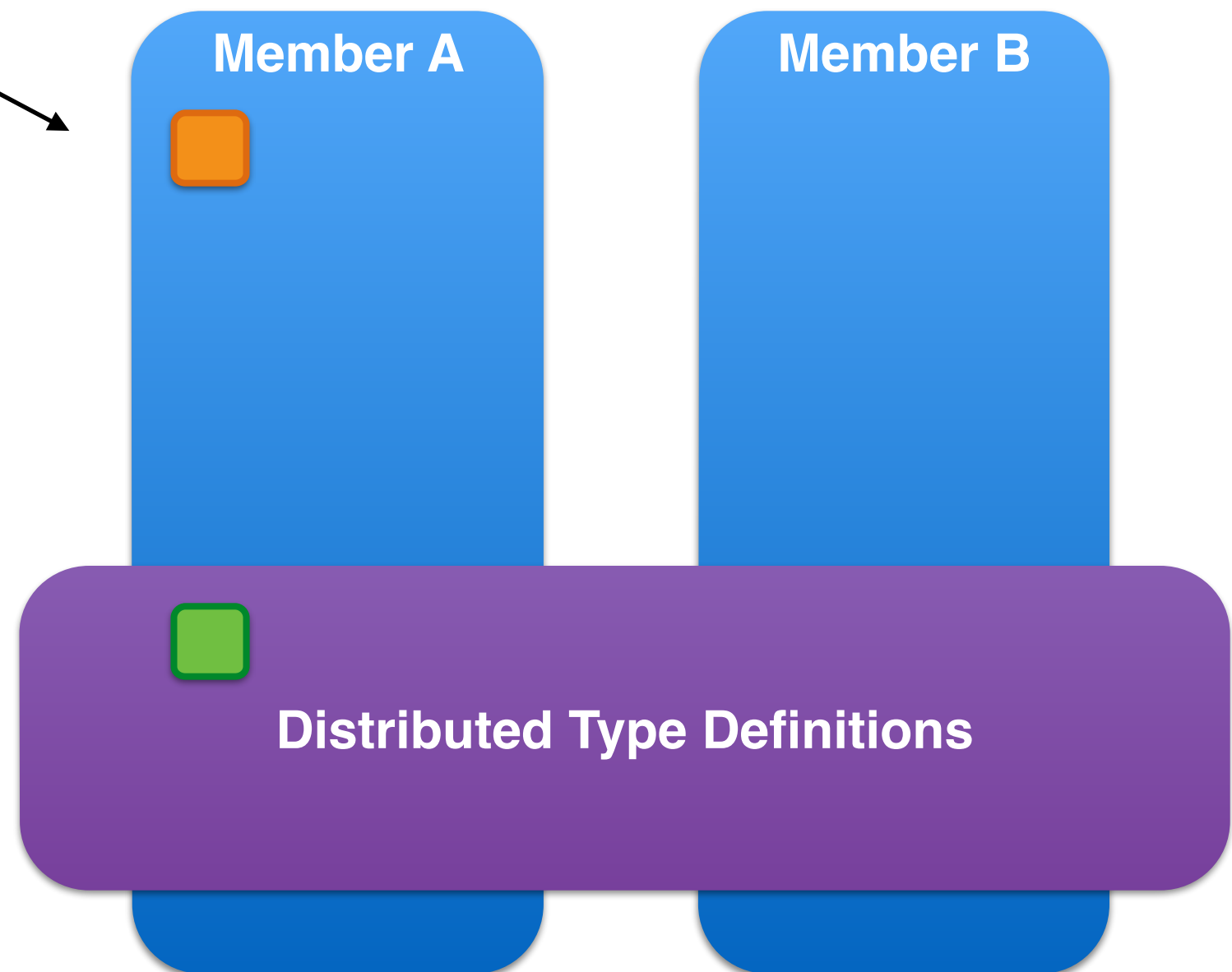
```
Person p1 = ...  
region.put("Fred", p1);
```



Distributed type registry

```
Person p1 = ...  
region.put("Fred", p1);
```

automatic definition

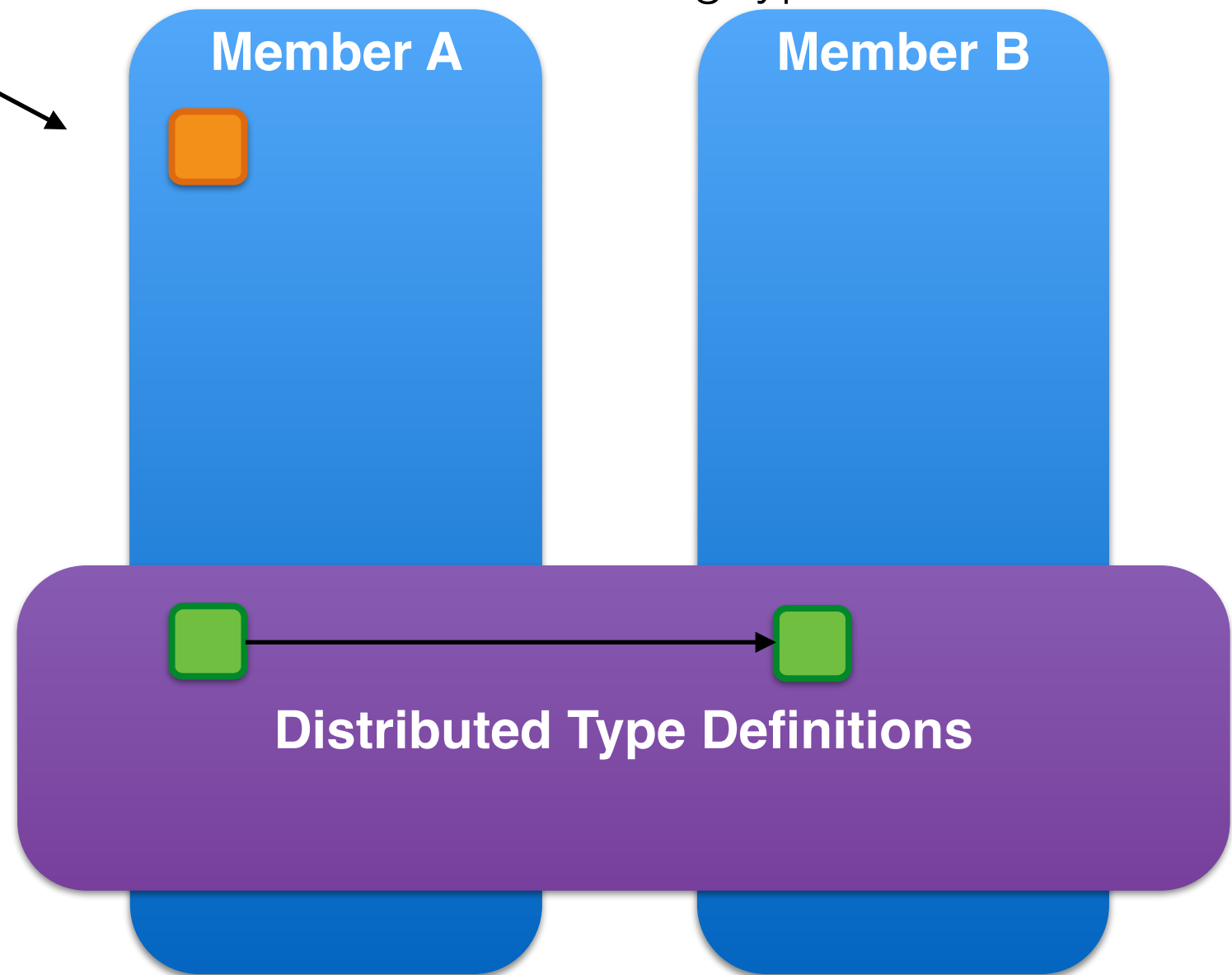


Distributed type registry

```
Person p1 = ...  
region.put("Fred", p1);
```

automatic definition

replicate serialized
data containing typeId

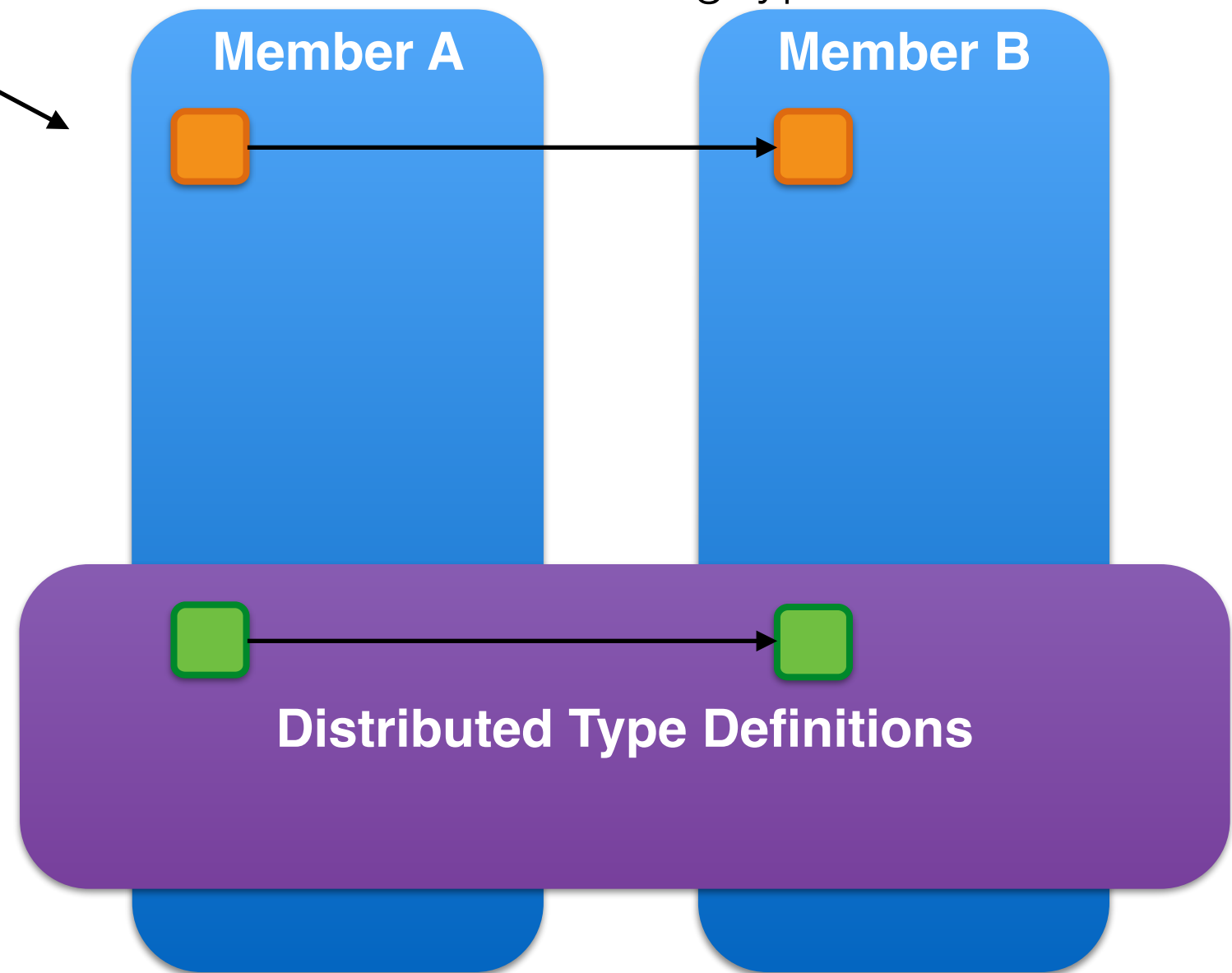


Distributed type registry

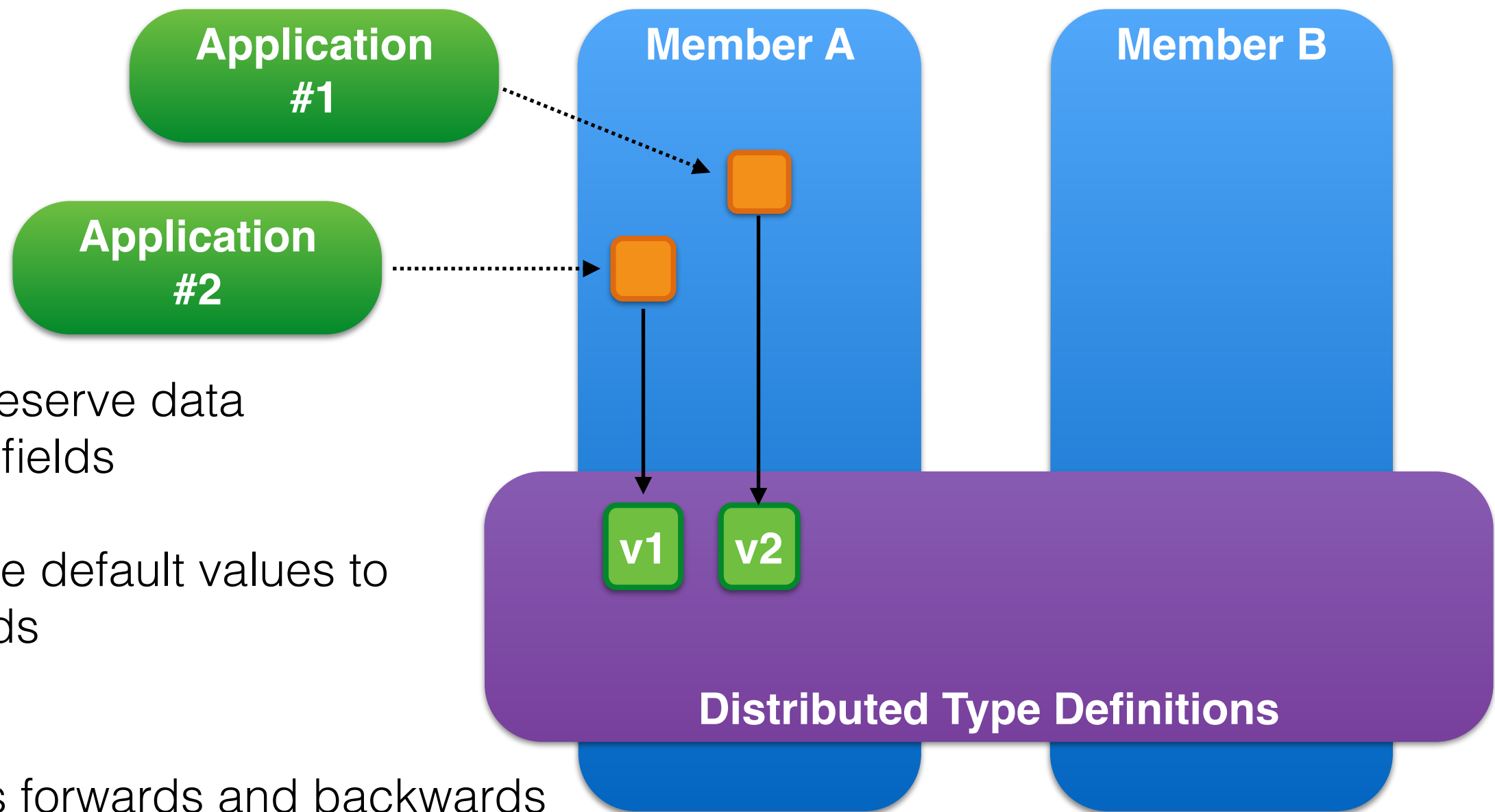
```
Person p1 = ...  
region.put("Fred", p1);
```

automatic definition

replicate serialized
data containing typeId



Schema evolution



v2 objects preserve data
from missing fields

v1 objects use default values to
fill in new fields

PDX provides forwards and backwards
compatibility, no code required

Show me the code!




Clone and build:

```
git clone https://github.com/apache/incubator-geode  
cd incubator-geode  
./gradlew build -Dskip.tests=true
```

Start a cluster:

```
cd gemfire-assembly/build/install/apache-geode  
./bin/gfsh  
gfsh> start locator --name=locator  
gfsh> start server --name=server  
gfsh> create region --name=myRegion --type=REPLICATE
```



The Geode Project

Why OSS? Why ASF?

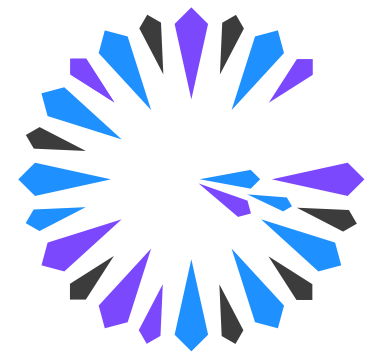
- Open source is fundamentally changing software buying patterns
- Customers avoid vendor lock-in and get transparency, co-development of features
- “Open source is where ecosystems are built”
- It's the community that matters
- ASF provides a framework for open source

Some context

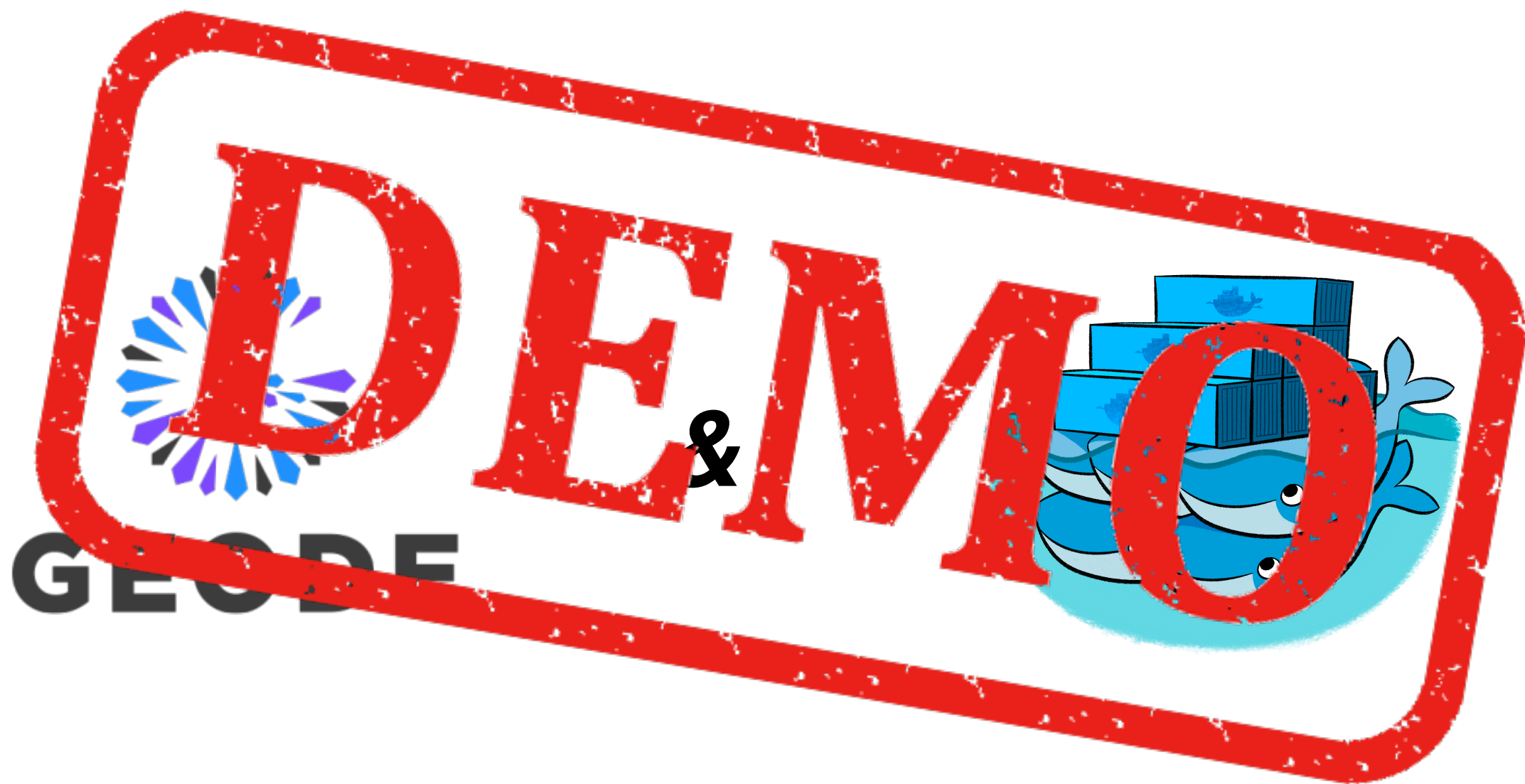
- 1M+ LOC, over a 1000 person-years invested into cutting edge R&D
- Thousands of production customers in very demanding verticals
- Cutting edge use cases that have shaped product design
- A core technology team that has stayed together since founding
- Performance differentiators baked into every aspect of the product

Geode vs GemFire

- Geode is project supported by the OSS community
- GemFire is a product from Pivotal, based on Geode source
- We donated everything but the kitchen sink*
- Development process follows “The Apache Way”



* Multi-site WAN replication, continuous queries, native (C++, .NET) client



GEODE

Active development

- Off-heap memory storage
- HDFS persistence
- Lucene indexes
- Spark connector
- Transactions on distributed data



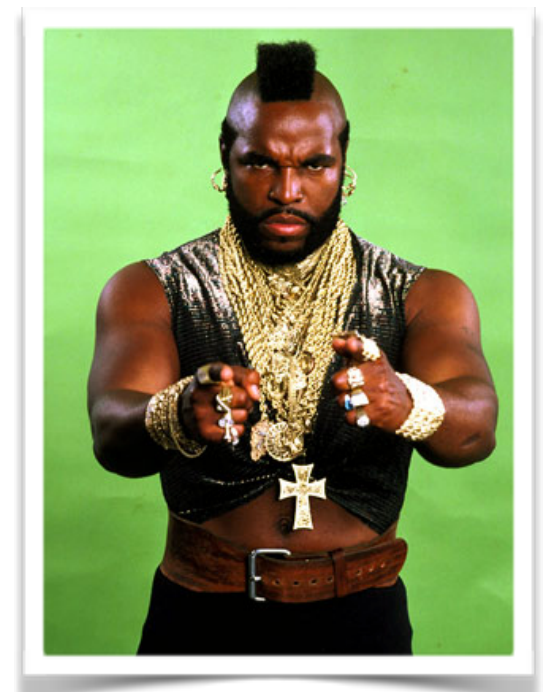
... and other ideas from the **Geode community!**

How to get involved

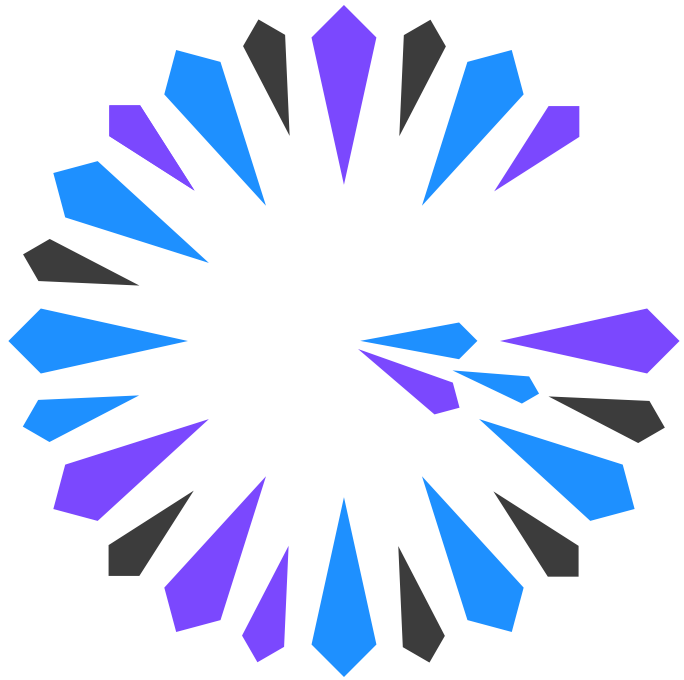
- Join the mailing lists; ask a question, answer a question, learn

dev@geode.incubator.apache.org
user@geode.incubator.apache.org

- File a bug in JIRA
- Update the wiki, website, or docs
- Create an example application
- Use it in your project!



We need you!



APACHE GEODE

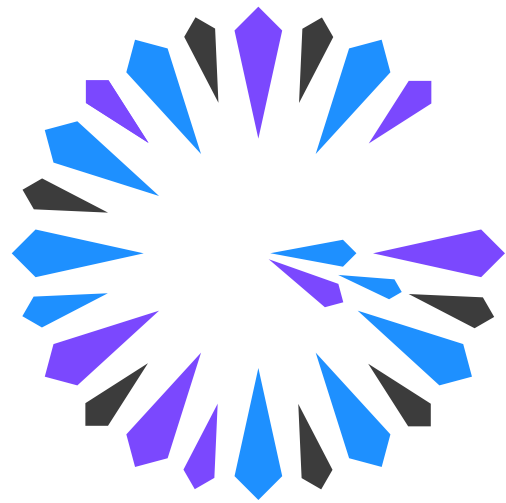
Thank you!

<http://geode.incubator.apache.org>



@ApacheGeode





APACHE GEODE

***Building a Highly-Scalable Open-Source Real-time Streaming Analytics System
Using Spark SQL, Apache Geode (incubating), SpringXD and Apache Zeppelin
(incubating)***

*Room: Tas - 15:00, Sep 29
Fred Melo, Pivotal, William Markito, Pivotal*

***Implementing a Highly Scalable In-Memory Stock Prediction System with Apache
Geode (incubating), R, SparkML and Spring XD***

*Room: Tohotom - 14:30, Sep 30
Fred Melo, Pivotal, William Markito, Pivotal*

