



Lightweight Grids with Terracotta

PT JUG, March 6, 2008

Cesario Ramos

Xebia

www.xebia.com



Goal of this session

- Understand how availability and scalability issues could be solved using a grid approach
- Understand how you could use Terracotta to create a lightweight grid in a simple manner.



Overview

- **Enterprise application challenges**
- How can a grid approach help?
- Terracotta Overview
- A simple grid example using Terracotta
- Q&A



Application challenges

- Availability

- Measured as the probability of being down.
- How do you keep your systems up?
 - Important to avoid a Single Point of Failure.

- Scalability

- Increase in computing power proportionally to the capacity added.
- How do you make your system scalable?
 - Important to avoid Single Point of Bottleneck.



High Availability

- Take into account that:
 - Redundancy in machinery means that:
 - 1 node: $p(\text{down}) = .01$ (99%)
 - 2 node: $p(\text{down}) = .01 * .01 = .0001$ (99.99%)
 - Adding tiers can decrease availability:
 - Three tiers with 99% uptime each gives $.99 * .99 * .99 = 97\%$ uptime.
 - Often better to have more 'small' than few 'big' machines.
 - Eight 1-cpu machines often better than two 4-cpu machines.
 - Node failure increases load on others.
- Focus on
 - Use buffers to protect tiers (failure)
 - Decouple Tiers (e.g. using JMS)
 - Use caching

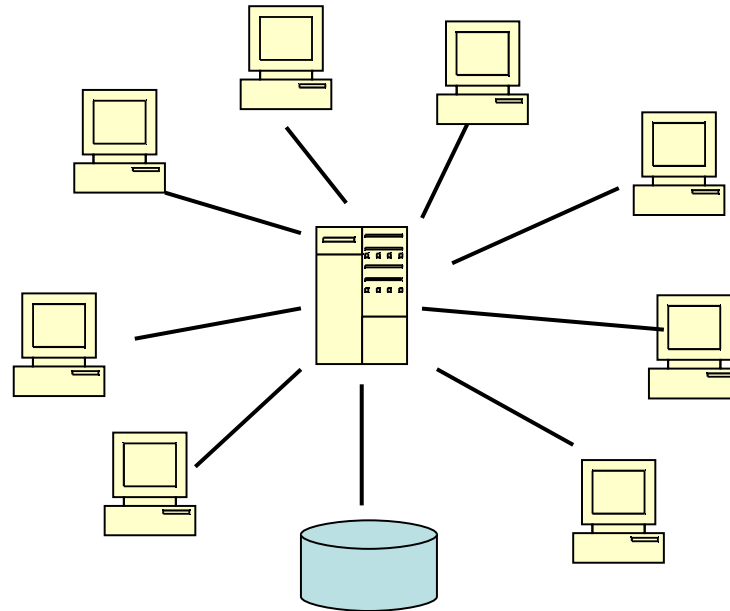


High Scalability

- Focus on:
 - Co-location of tiers.
 - Reduce cost of inter tier communication
 - Reduce cost of serialization and remote objects
 - Avoid concurrency control bottlenecks.
 - Isolation levels, application lock contention, ...
 - Favor bottleneck in the application tier.
 - CPU, memory bound.
 - Use caching for
 - Fast data access
 - Unloading of expensive tiers



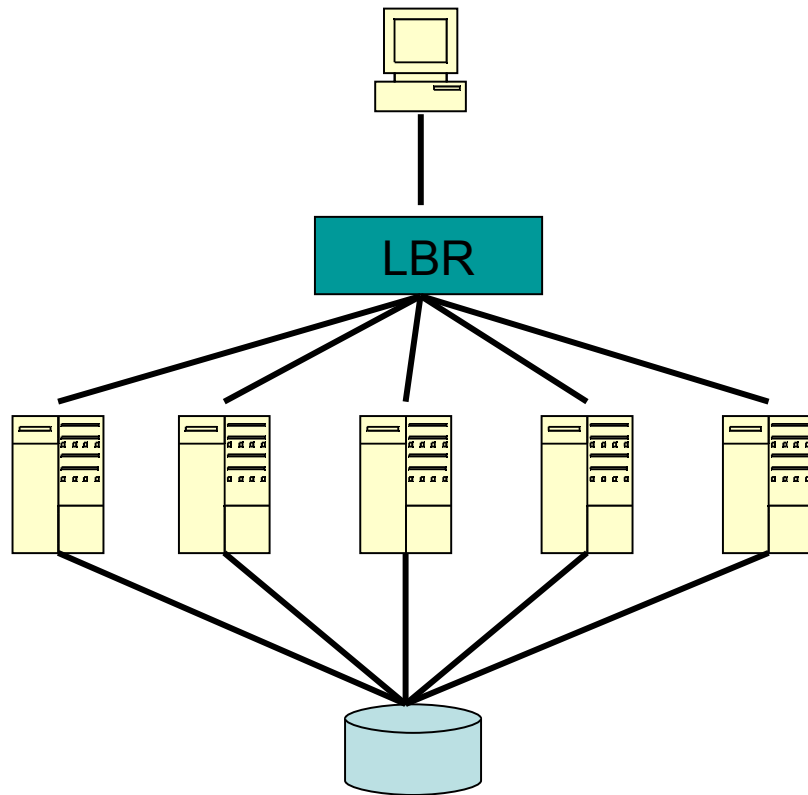
The client/server architecture



- Single Point Of Failure.
- Single Point Of Bottleneck.



Clustering your application



- Cluster the servers.
 - Use Serialization, RMI, JMS, ...
- Keep session state small, max 10kb.
- Store state in db.
 - Transient state.

- Unload responsibility to the database
 - Complicated and expensive to cluster.



Summary

- Try to avoid SPOF and SPOB.
- Don't misuse your EIS for transient data.
 - Eventually it will become a SPOB
- Often better to have more 'small' than few 'big' machines.
- Use caching to unload expensive tiers.

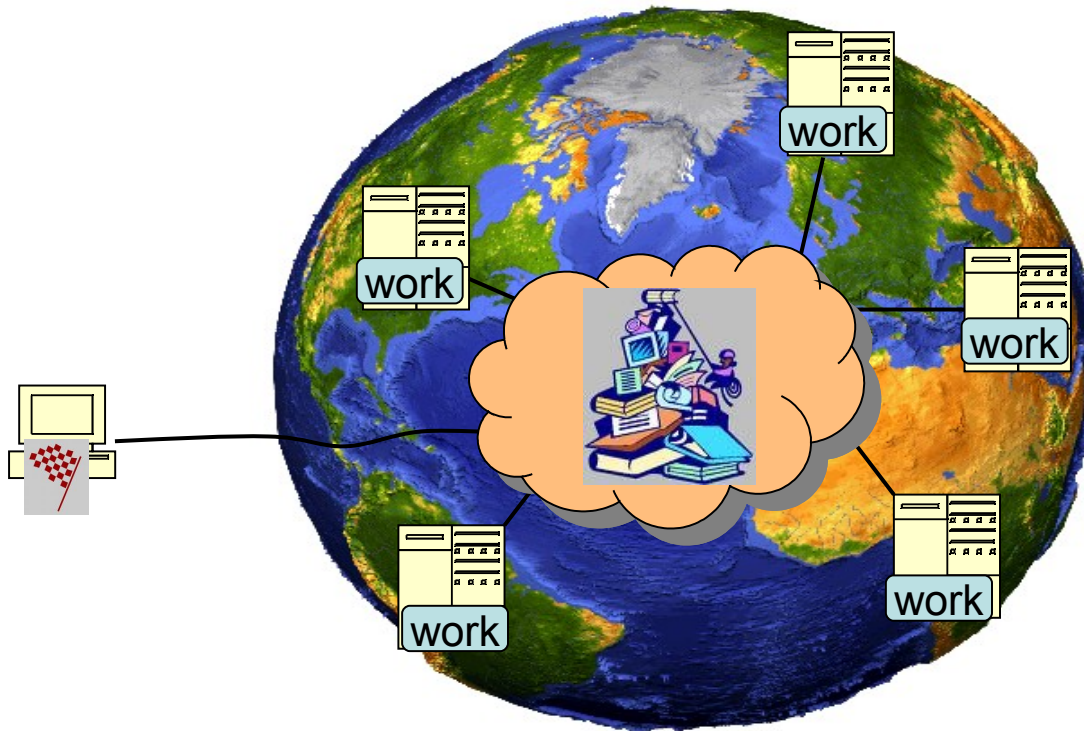


Overview

- Enterprise application challenges.
- **How can a grid approach help?**
- Terracotta Overview
- A simple grid example using Terracotta.
- Q&A.



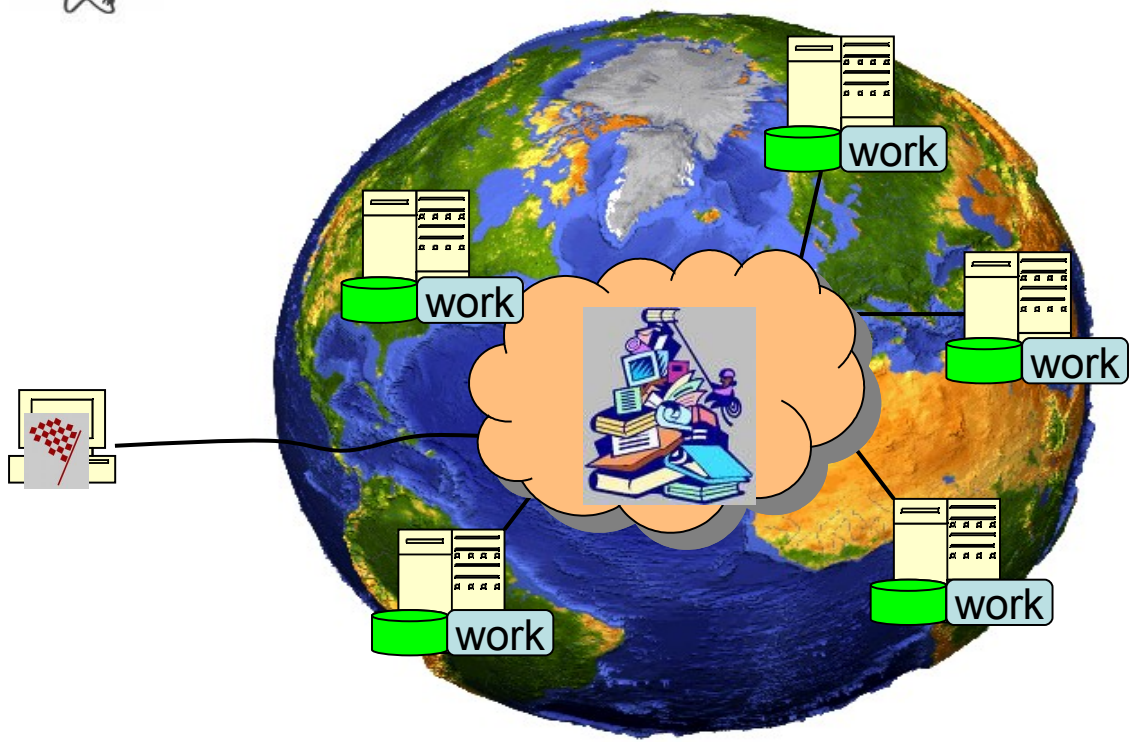
What is a computational grid?



Increased computing power is bounded by its data access speed



What is a data grid?



- Locality of reference.



Caching topologies

- Partitioned
 - Partition data across all members
 - High scalability
 - Load balanced
 - Fits nicely on a data grid
- Replicated
 - Replicate data to all members
 - High Performance
 - Limits on data update and entry



Summary

- Availability
 - Use of data duplication
 - Redundancy in machinery and replication of components
- Scalability
 - Use of Locality of Reference
 - Move operations around instead of data
 - Dynamic allocation of resources.
- Grid middleware takes care of
 - Distribution of load, logic, data and events.
 - Dynamically adopt new nodes
 - Persistence.
 - Transactions.



Overview

- Enterprise application challenges.
- How can a grid approach help?
- **Terracotta Overview.**
- A grid example using Terracotta.
- Q&A.

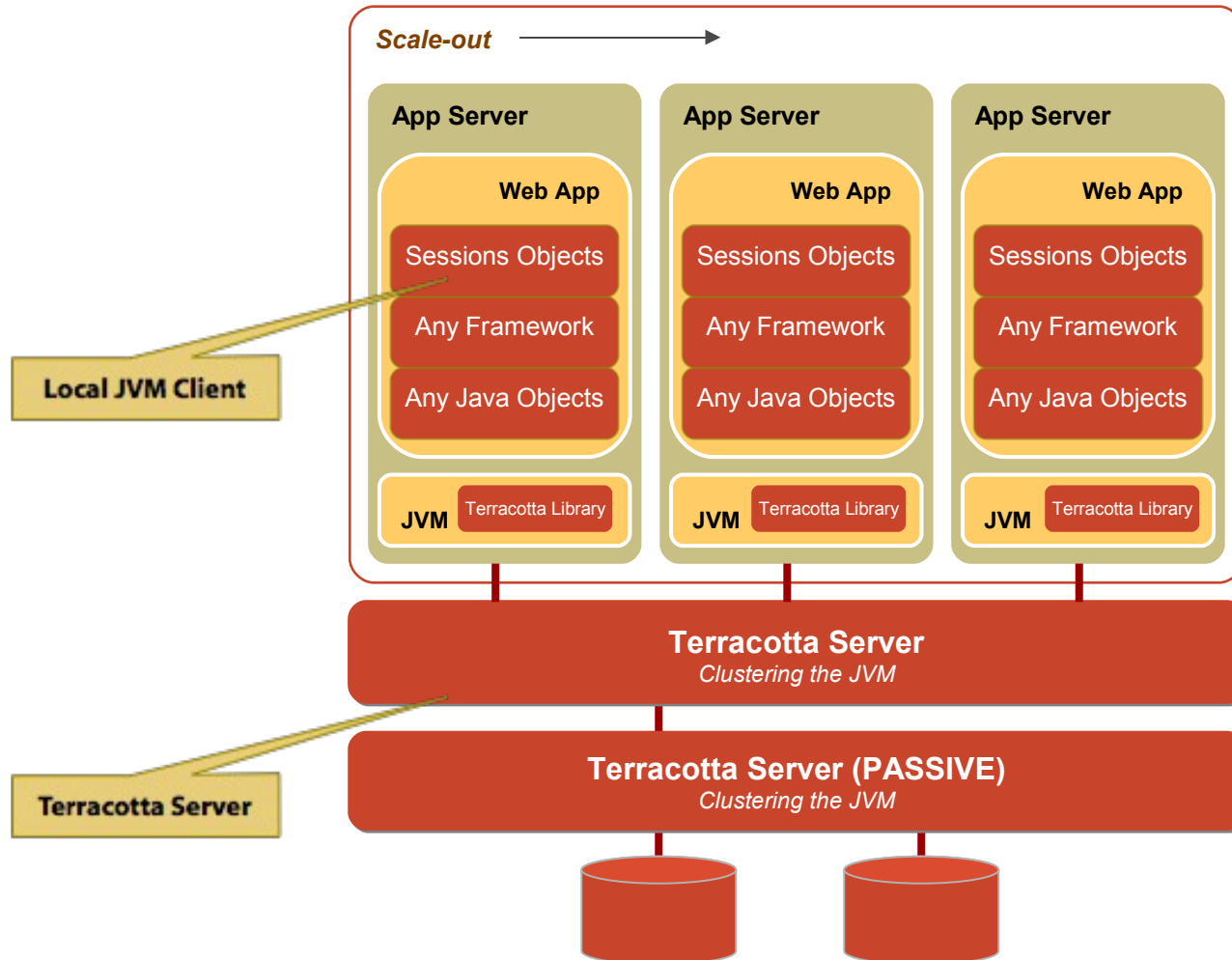


Terracotta in a nutshell?

- Open Source clustering solution for Java
- Availability and scalability at the JVM level
- Allows you to write Java applications for a single JVM and have it distributed over various JVMs in a transparent way
- Reduces the complexity of distributed computing
- Provides a basis for realizing grid solutions
- Runtime monitoring and control
- No API



Terracotta architecture





How to use the extended heap?

- Distributed Shared Object.
 - Terracotta instrumented object.
 - Cluster wide unique object.
- Shared root
 - starting point of a distributed object graph
 - class data member that is transparently mirrored
 - are persisted across JVM lifecycle
- Terracotta transaction
 - MONITOR ENTRY and MONITOR EXIT
 - Changed data is synchronized using synchronous commit.
- Locks
 - Cluster wide synchronized, wait/notify
 - Synchronous-Write, Write, Read, Concurrent
 - Named, Auto



What does Terracotta offer?

- Availability and Scalability
 - Data written to TC is persistent until explicitly removed
 - Redundant TC servers.
 - Fine grained changes. (No serialization)
 - Copy only where resident.
 - Automatic partitioning.
- Dynamic allocation of resources.
- Transparent distribution and partitioning of
 - logic, data or events.
- Transactions.
 - Java memory model semantics
 - Lock acquire and lock release for flushing data to TC.



Simple Example

```
public class Consumer {
private  LinkedBlockingQueue<Something>
    queue;
public void consume() {
    Something msg = null;
    while(true) {
    msg = queue.take()
...
public class Producer {
private  LinkedBlockingQueue<Something>
    queue;
public void produce() {
    Something msg = null;
    while(true) {
    msg = queue.put();
...

```

tc-config.xml

```
...
<instrumented-classes>
  <include>
    <class-expression>
      example.Something
    </class-expression>
  ...
<root><field-name>
  example.Consumer.queue
</field-name>
<root-name>rootQueue</root-name>
</root>
<root>
  <field-name>Producer.queue</field-name>
  <root-name>rootQueue</root-name>
</root>
...

```



Terracotta use cases

- Grid solutions
 - E.g. using WorkManager spec by IBM & BEA
- Clustering POJOs
- Distributed caching
 - E.g. standalone cache, Hibernate L2
- Clustering Spring
- HttpSession clustering
 - No serialization, No SetAttribute(...)



Summary

- Availability and scalability at the JVM level.
- Keeps JVM semantics across the cluster.
- Reduces complexity of distributed computing.
- Provides a basis for realizing grid solutions.

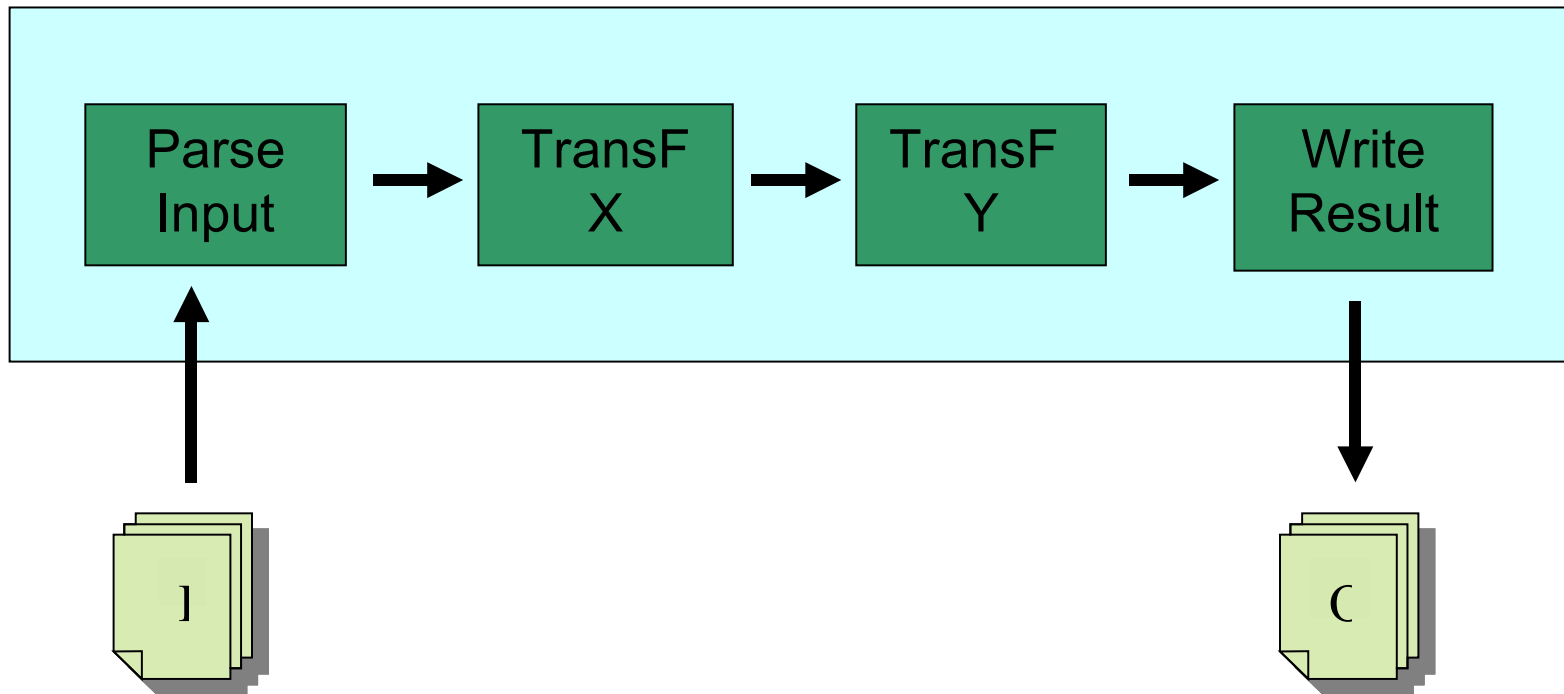


Overview

- Enterprise application challenges.
- How can a grid approach help?
- Terracotta Overview.
- **A simple grid example using Terracotta.**
- Q&A.



Use Case



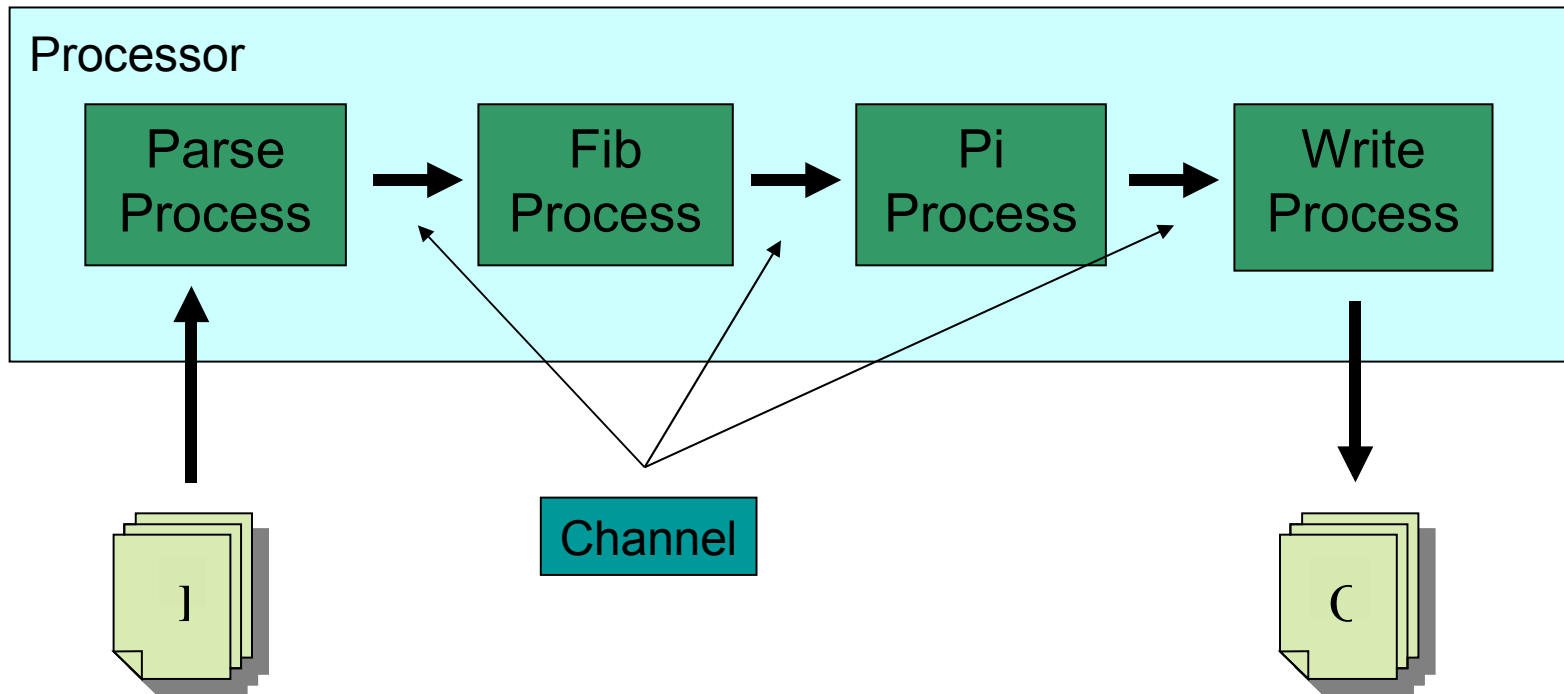


Prometheus

- Library that extends `java.util.concurrent`.
- Makes it easy to separate business logic from concurrent infrastructure logic.
- Offers coarse grained building blocks i.e.
 - Processor
 - Process
 - Repeater
 - Channel



Fits nicely in Prometheus





Fibonacci Process

```
public class FibonacciProcess {  
    ...  
    public void receive(Task task) {  
        task.setOutFibonacci(fibonacci(task.getInFibonacci()));  
        log.info(task);  
    }  
    public static long fibonacci(long n) {  
        if (n <= 1)  
            return n;  
        else  
            return fibonacci(n - 1) + fibonacci(n - 2);  
    }  
}
```



Spring config: tc-config

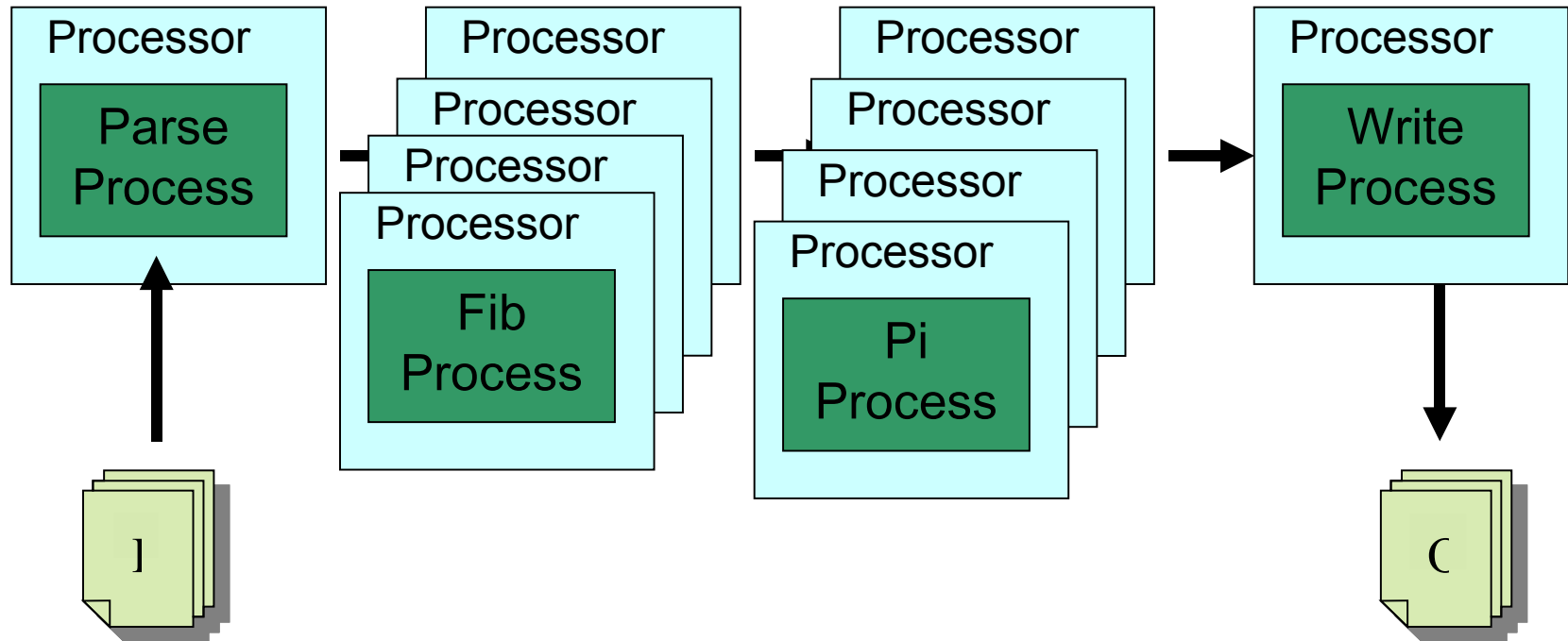
...

```
<application-contexts>
  <application-context>
    <paths>
      <path>applicationcontext-channels.xml</path>
    </paths>
    <beans>
      <bean name="parseToFib"/>
      <bean name="FibToPi"/>
      <bean name="PiToWrite"/>
    </beans>
  </application-context>
</application-contexts>
```

...



Distributed parallelized





Wrapping up

- Terracotta turns scalability and high-availability into a **deployment artifact**
- Keep the simplicity of POJO-based development – get **Scale-Out with Simplicity**
- Makes mission-critical applications **simpler to:**
 - Write
 - Understand
 - Maintain
 - Test
- Endless possibilities for clustering and distributed programming - these were just a few
- Be creative, use your imagination and have fun...
- But beware, TC introduces subtle changes:
 - Semantics: tc-roots
 - API's and behaviour of modules like EHCache



Final Note

- Want to hear more?
 - blogs.xebia.com
 - podcast.xebia.com
 - Interview with Jonas Boner from Terracotta
- Want to learn more about performance?
 - Xebia course on may 2008:
 - Speeding up Java applications of Kirk Pepperdine
- Prometheus
 - <http://prometheus.codehaus.org/>



Lightweight Grids with Terracotta

Cesario Ramos

Xebia