Non-functional benefits of Scaling your Web Tier using Coherence*Web

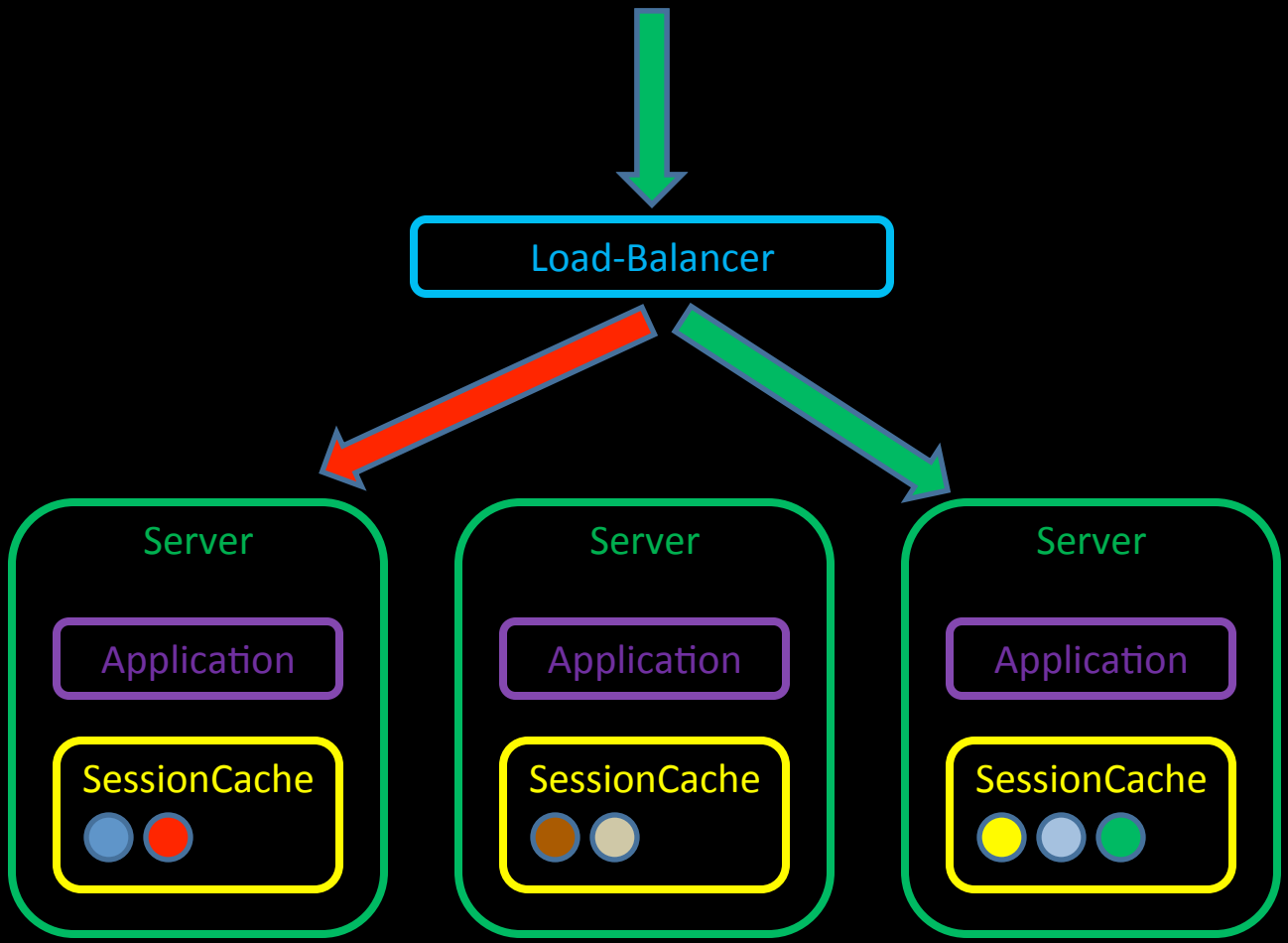
Mark Addy, Consultant

Fast, Reliable, Manageable & Secure

HTTP Session Management

Life Before Coherence

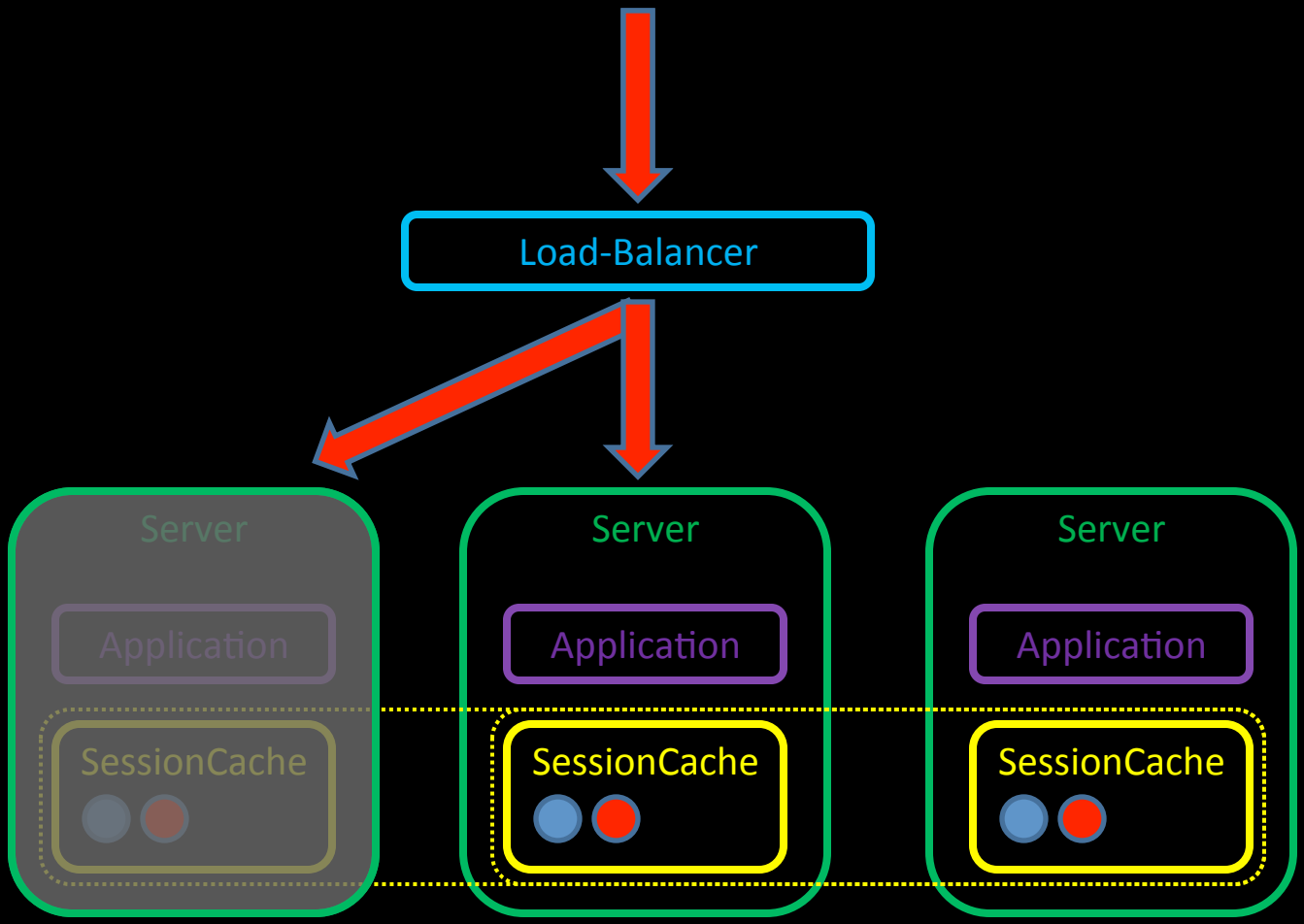
Non-Clustered



Non-Clustered

- Sticky Session Load-Balancer
- No Redundancy
- Session Cache bloats Application JVM Heap

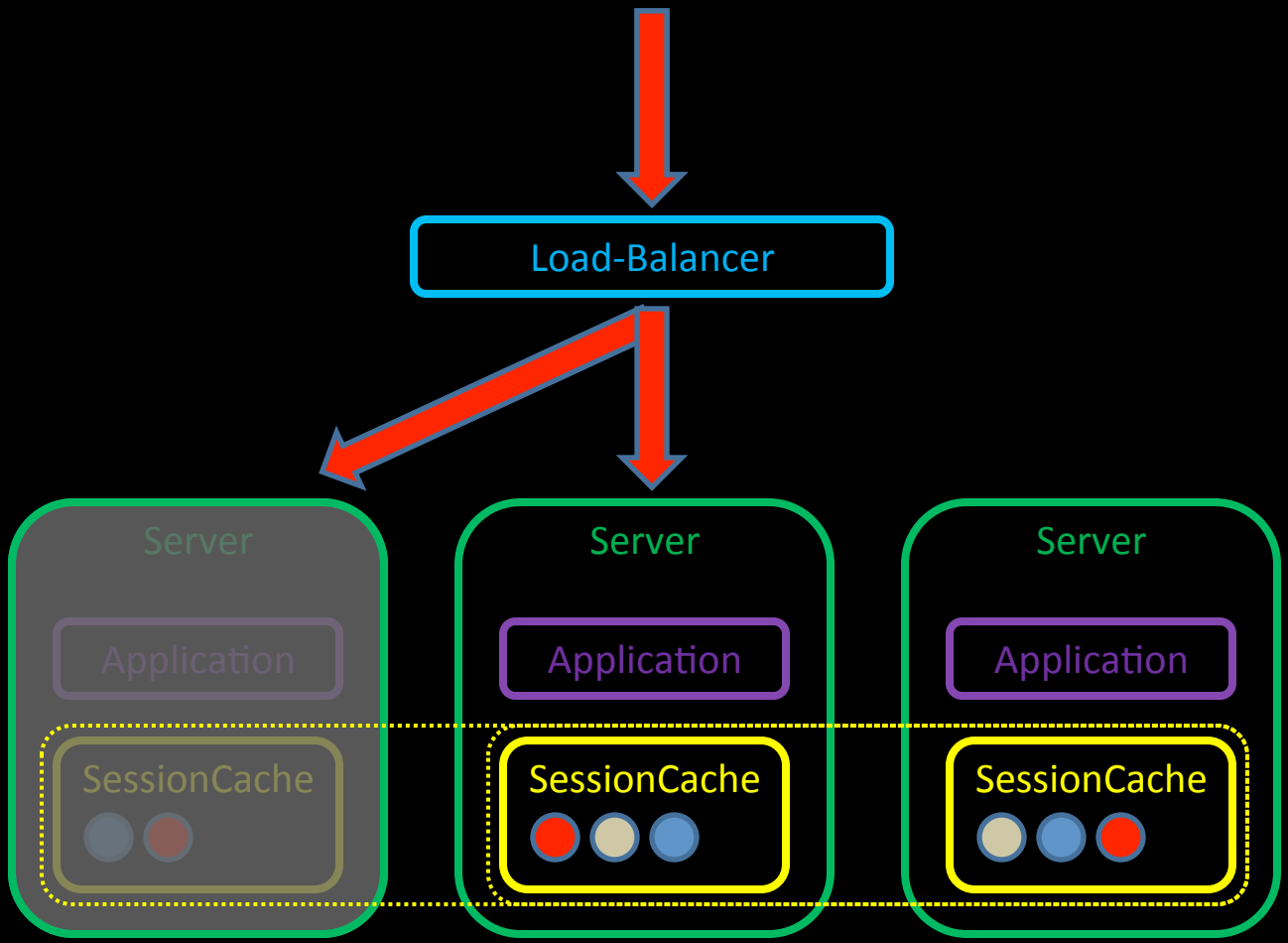
Clustered - Replicated



Clustered – Replicated

- Sticky Session Load-Balancer
- Redundancy / Failover
- Too much Redundancy?
- Session Cache bloats Application JVM Heap
- Replication overhead increases proportionally to the number of nodes
- Adding more node != more Session Storage capacity
- Tuning and Scaling of the Session Cache is coupled to the Application

Clustered – Backup / Buddy Replicas

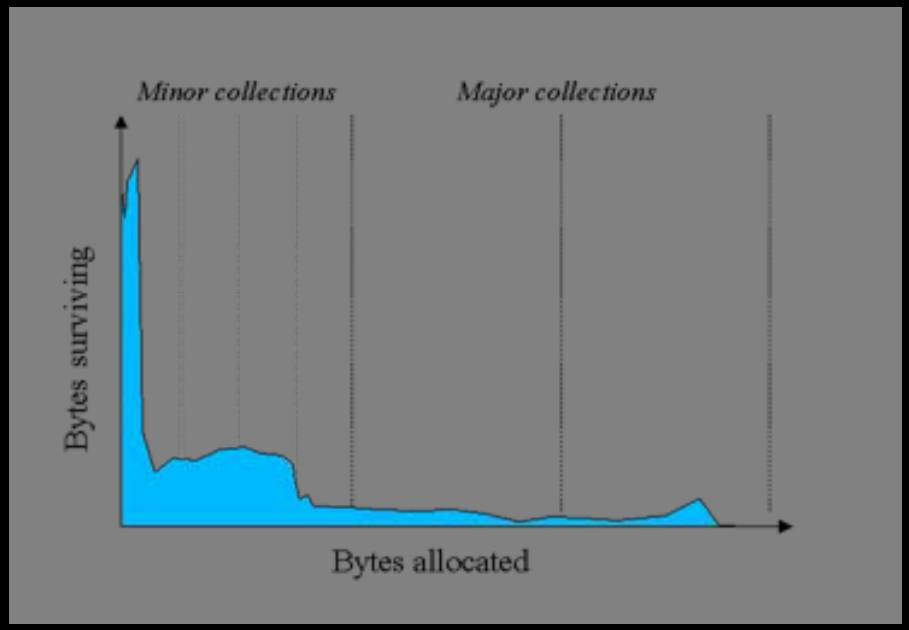


Clustered – “Buddy Replicas”

- Sticky Session Load-Balancer
- Implementations
  - WebLogic Clustering
  - JBossCache Buddy Replication
  - Tomcat Backup Manager
- Redundancy / Failover
- Redundancy is more realistic
- Adding more nodes == Increased session storage
- Session Cache bloats Application JVM Heap
- Tuning and Scaling of the Session Cache is coupled to the Application

The Challenges

- Redundancy
  - Transparent Failover
  - Rolling upgrades / zero downtime
- Coupling
  - Independent Tuning
  - Independent Scaling
- Latency
  - High access HttpSession Metadata / Attributes



Coherence*Web

Coherence*Web

- Http Session Management Module
- Built directly on top of Coherence
  - Scalability
  - Availability
  - Reliability
  - Performance

Coherence*Web Integration

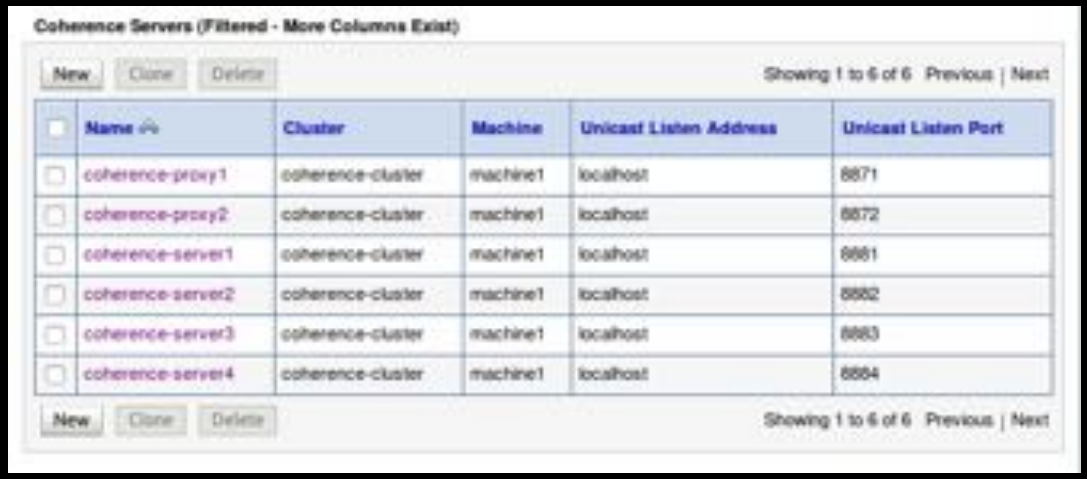
Active Cache

Integration into WebLogic Server & GlassFish

- WebLogic 10.3.3+
  - Requires installation of active-cache and coherence-web-spi shared libraries

Active Cache

- Management via console or WLST
- Process control via Node Manager
  - Including auto-restart for failed nodes
- Application Configuration via WebLogic Deployment Descriptors



Coherence Servers (Filtered - More Columns Exist)

New | Clone | Delete — Showing 1 to 6 of 6 Previous | Next

Name	Cluster	Machine	Unicast Listen Address	Unicast Listen Port
coherence-proxy1	coherence-cluster	machine1	localhost	8871
coherence-proxy2	coherence-cluster	machine1	localhost	8872
coherence-server1	coherence-cluster	machine1	localhost	8881
coherence-server2	coherence-cluster	machine1	localhost	8882
coherence-server3	coherence-cluster	machine1	localhost	8883
coherence-server4	coherence-cluster	machine1	localhost	8884

New | Clone | Delete — Showing 1 to 6 of 6 Previous | Next

Other Containers

Supported

- Apache Tomcat 5.5.n, 6.0.n
- IBM WebSphere 5, 6, 7
- JBoss Application Server
- Jetty 5.1.n, 6.1n
- Oracle OC4J 10.1.2.n, 10.1.3.n
- Oracle WebLogic 9, 10
- Sun Application Server
- Sun GlassFish 2.n

Stage 1 - Inspection

```
java -jar webInstaller.jar ${war} -inspect -server:Tomcat/6.0.x
```

- Generates Coherence*Web parameters for target container (coherence-web.xml)
- Amend settings in this file prior to the installation stage

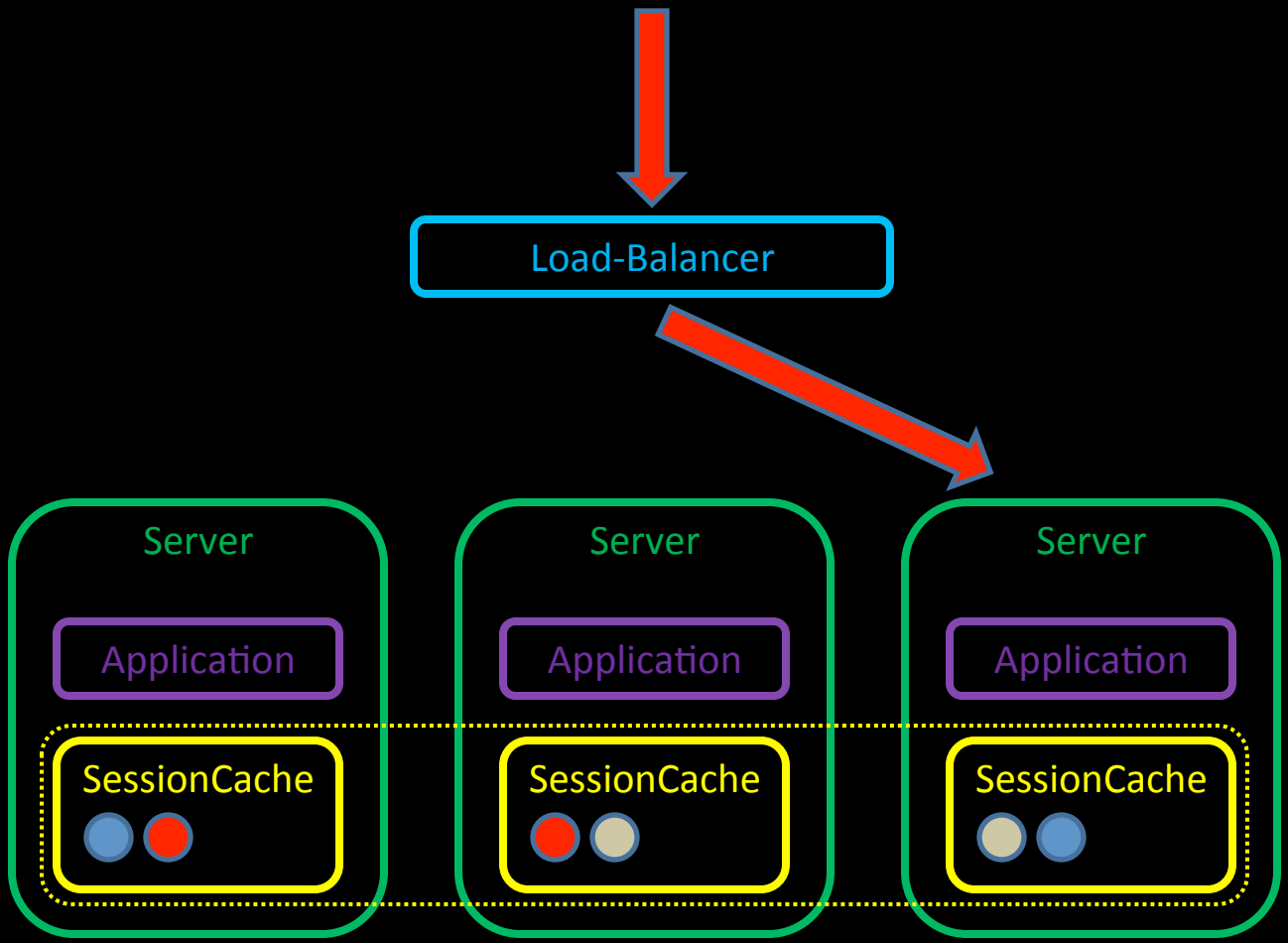
Stage 2 - Installation / Instrumentation

```
java -jar webInstaller.jar ${war} -install
```

- Applies Coherence*Web context parameters to Application web.xml using the coherence-web.xml file generated in the inspection step
- Unregisters Application ServletContext, ServletRequest and HttpSession Listeners
- Registers Coherence*Web ServletContext Listeners
- Wraps Servlets with Coherence*Web SessionServlet
- Wraps JSP's with Coherence*Web JspServlet

Redundancy, Availability & Scalability

Partitioned / Distributed Clustering



Partitioned / Distributed Clustering

- Configurable Number of back-ups
- Coherence Grid hides data-location
- Dynamic Scaling
- Replication overhead does not increase with cluster size
- Sticky Sessions might not be routed to the Node owning the data

The Coupling Challenge

Coherence*Web Topology

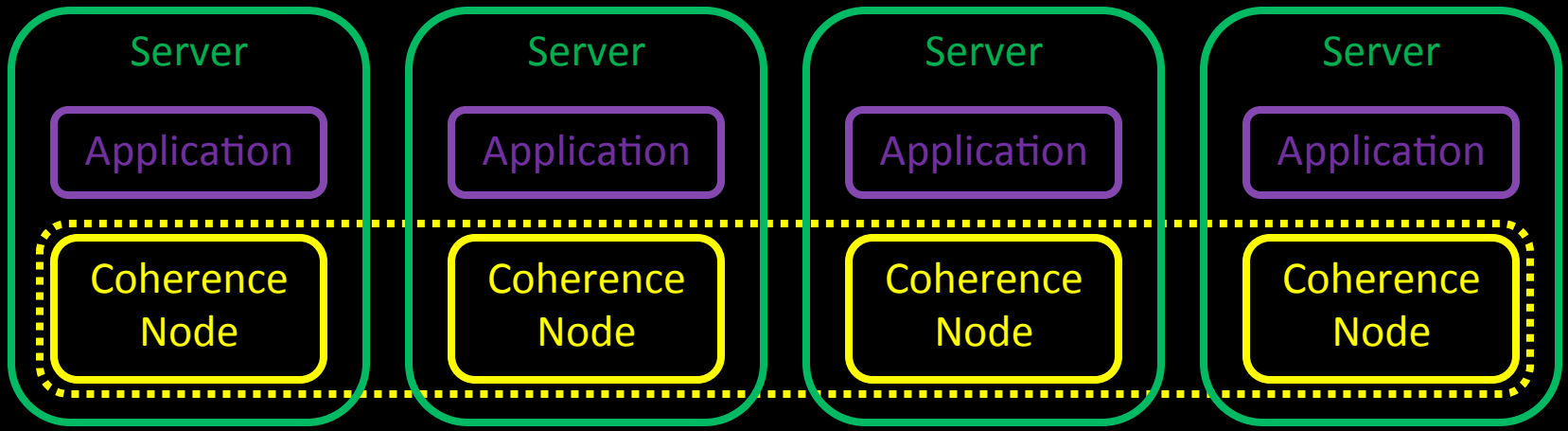
Traditional Http Session Caches are embedded

We have a choice to make:

- In-process
  - Embedded
- Out of process
  - Storage disabled
- Out of process
  - Coherence*Extend

In-Process

In-Process

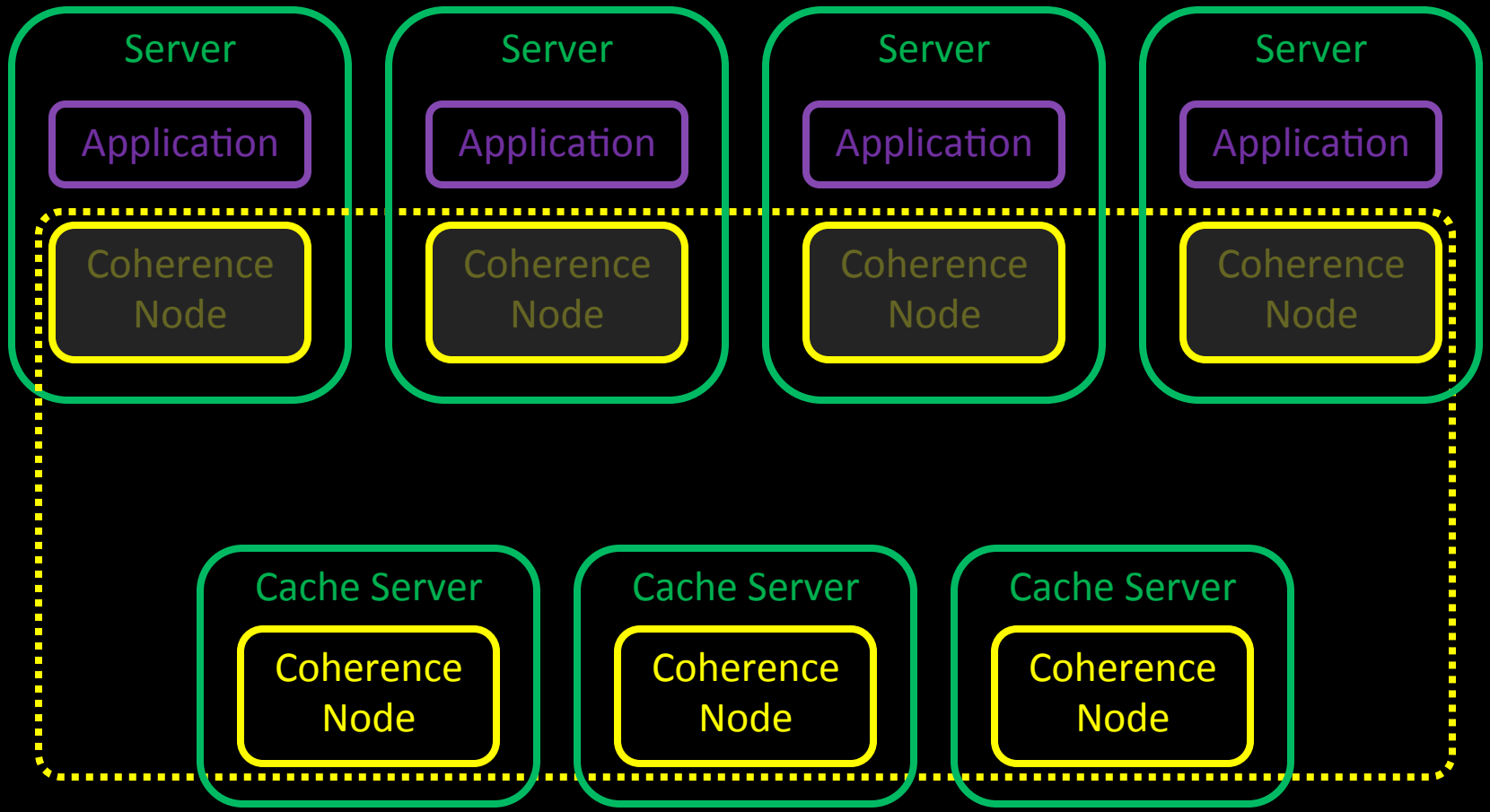

Server
Application
Coherence Node
Server
Application
Coherence Node
Server
Application
Coherence Node
Server
Application
Coherence Node

In Process

- Fast In memory access when replicated
- Low session numbers
- Good for Development
- Application shares same JVM as Cache
  - Unable to Scale independently
  - Unable to Tune Application and Cache independently
  - Restarts of Application Nodes impact the cluster
  - Increased Heap size and GC pauses

Out of Process Storage Disabled

Out of Process Storage Disabled



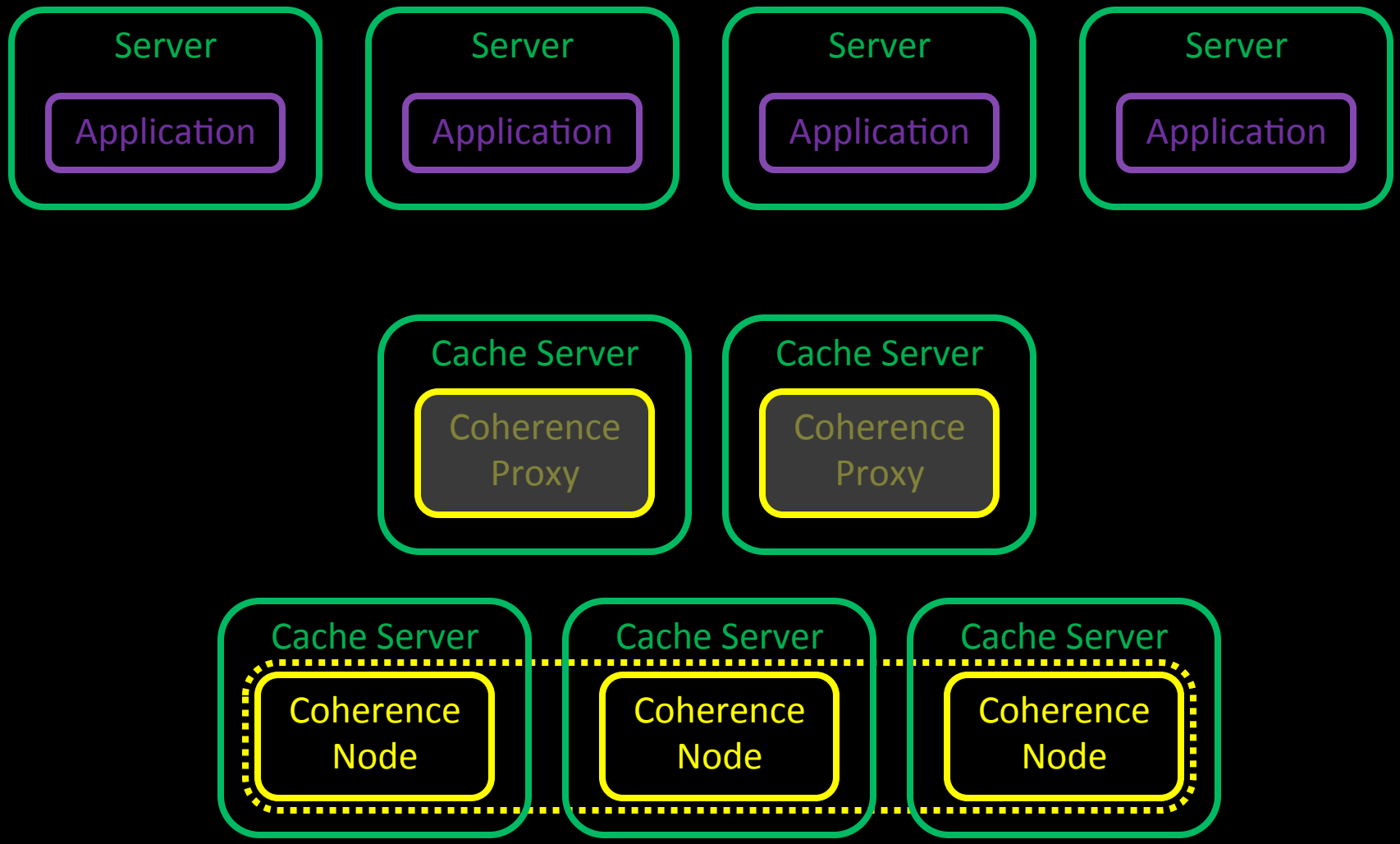
Out of Process Storage Disabled

- Straight forward to configure

  -Dtangosol.distributed.local.storage=false
- Application and Cache are separated
  - Independent tuning
  - Independent scaling
  - Looser Coupling
- Application and Cache Tiers are Clustered

Out of Process Coherence*Extend

Out of Process Coherence*Extend

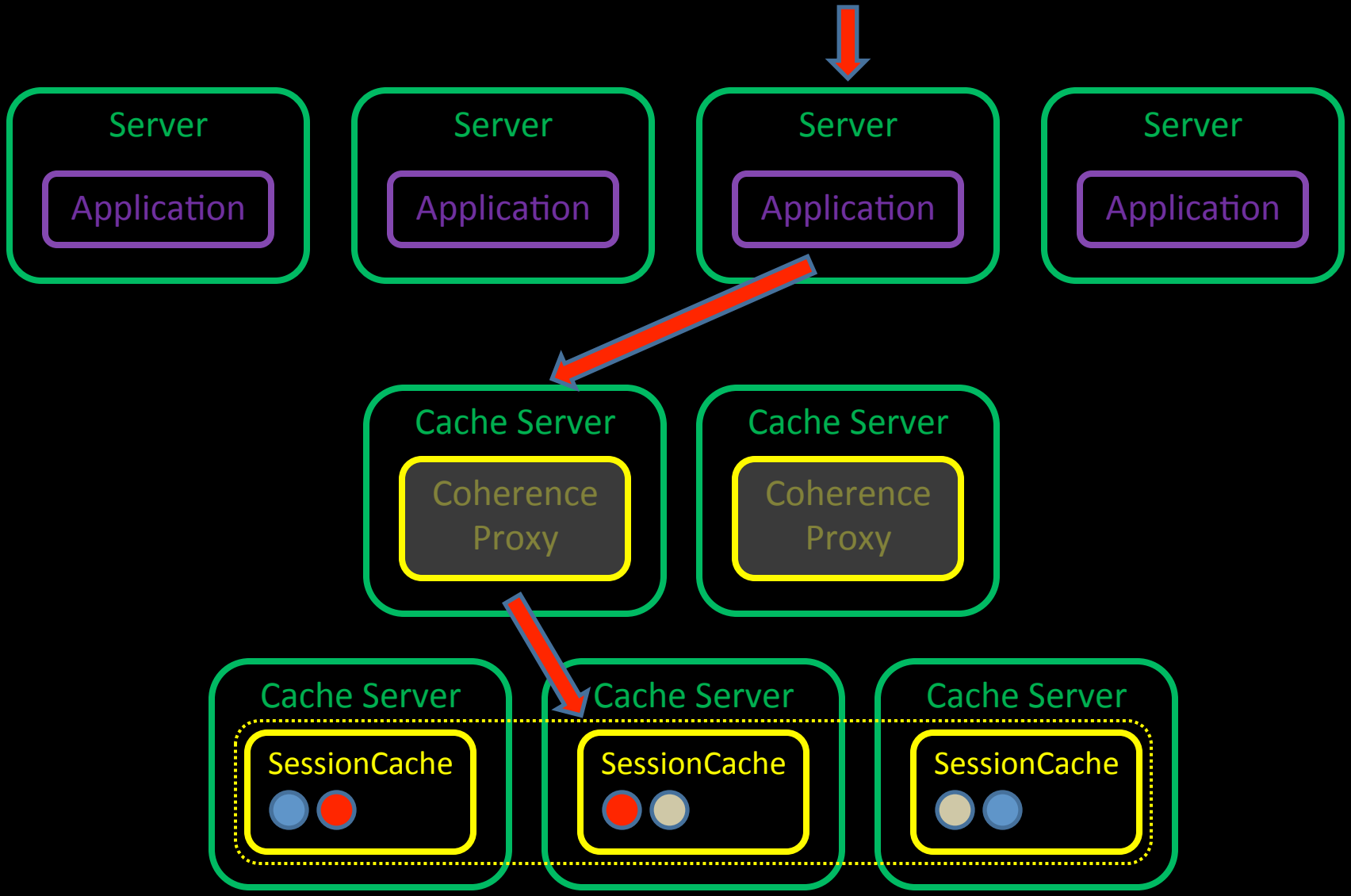


Out of Process Coherence*Extend

- Application and Cache are totally separated
  - Independent tuning
    - Throughput Collector for Application?
    - Low Pause Collector for Cache?
  - Independent scaling
  - De-coupled
  - Reduced memory requirement for Application JVM
  - Sharing Session Cache is possible
  - Increased Latency

The Latency Challenge

Coherence*Extend Latency

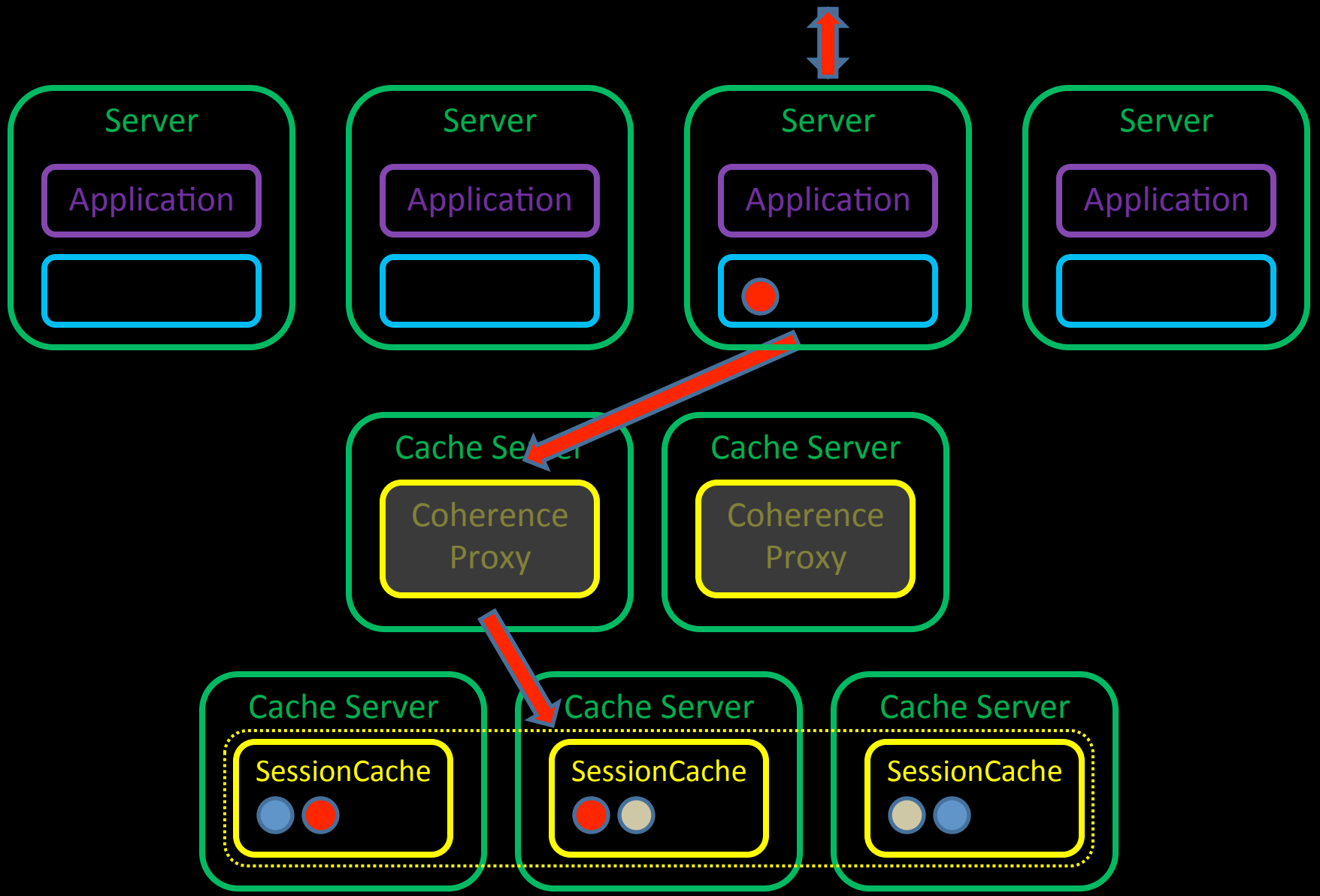


Reducing Latency

We have control over the Coherence Cache Configuration

```
<near-scheme>
  <scheme-name>near-cache-scheme</scheme-name>
  <front-scheme>
    <local-scheme>
      <scheme-ref>limited-local-near-cache</scheme-ref>
    </local-scheme>
  </front-scheme>
  <back-scheme>
    <remote-cache-scheme>
      <scheme-ref>unlimited-partitioned-remote-cache</scheme-ref>
    </remote-cache-scheme>
  </back-scheme>
</near-scheme>
```

Coherence*Extend Latency



Near Cache

- Front scheme is a subset of the Back scheme
- Local caching for high access cache entries
- Reduced Latency
- Brings session state close for Sticky Sessions
- Size limited to ensure Application heap is kept under control

Latency & The Session Model

Session Models

The Http Session

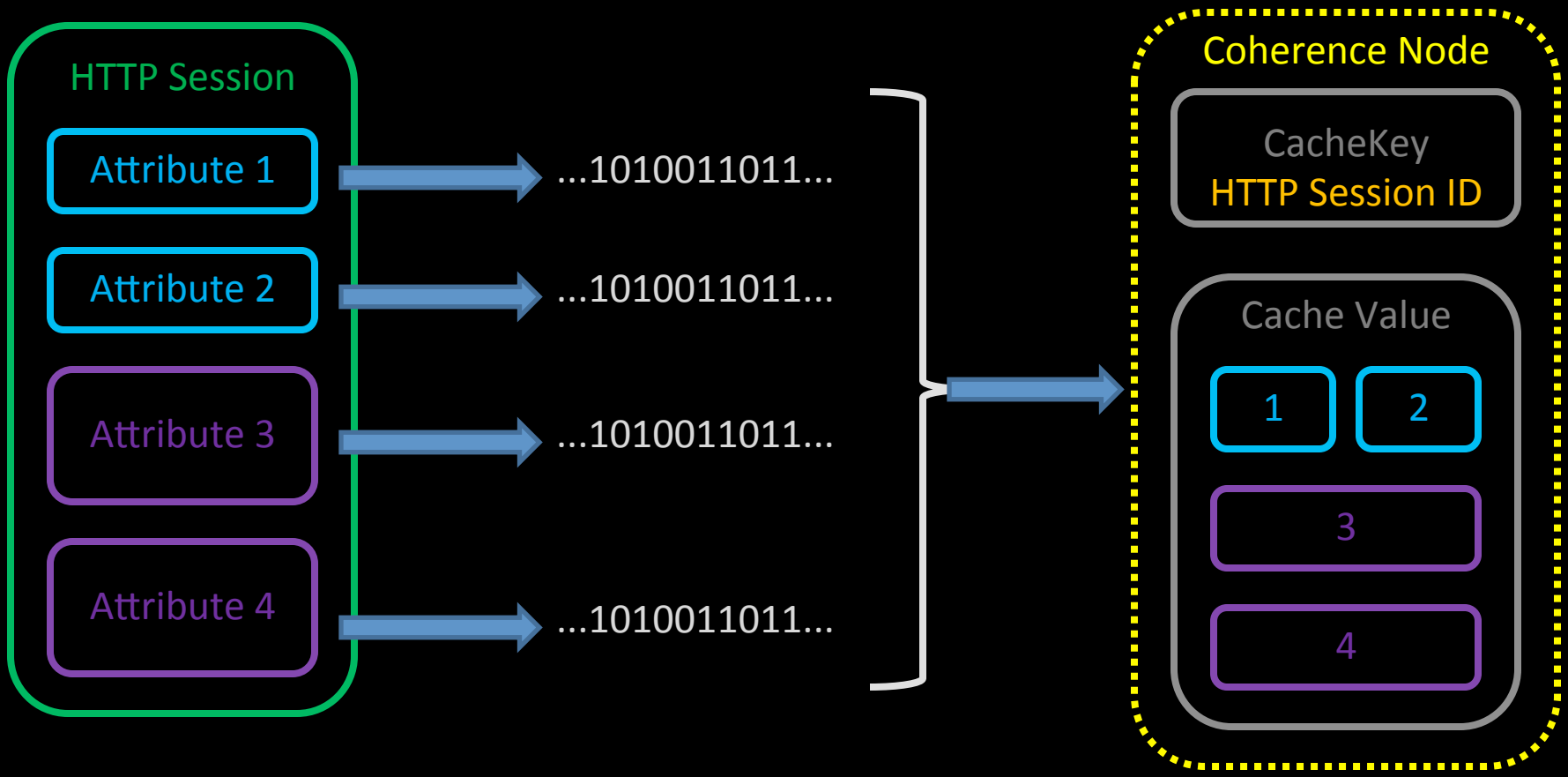
- MetaData
  - Last Accessed Time
  - Created Time
  - ID
- Attributes
  - Store what you want!
  - Not all Attributes are equal...

Coherence*Web Session Models

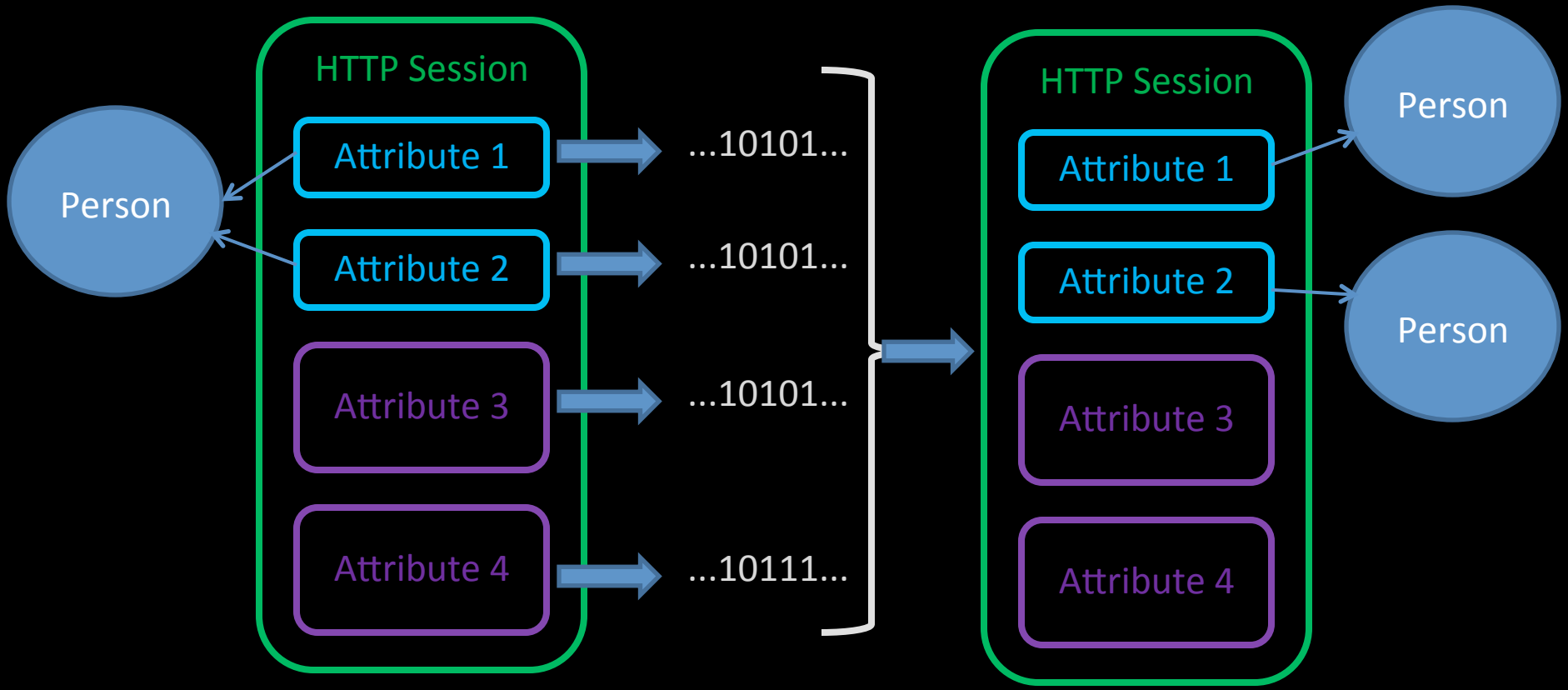
- Traditional
- Monolithic
- Split

Traditional

Traditional



Traditional

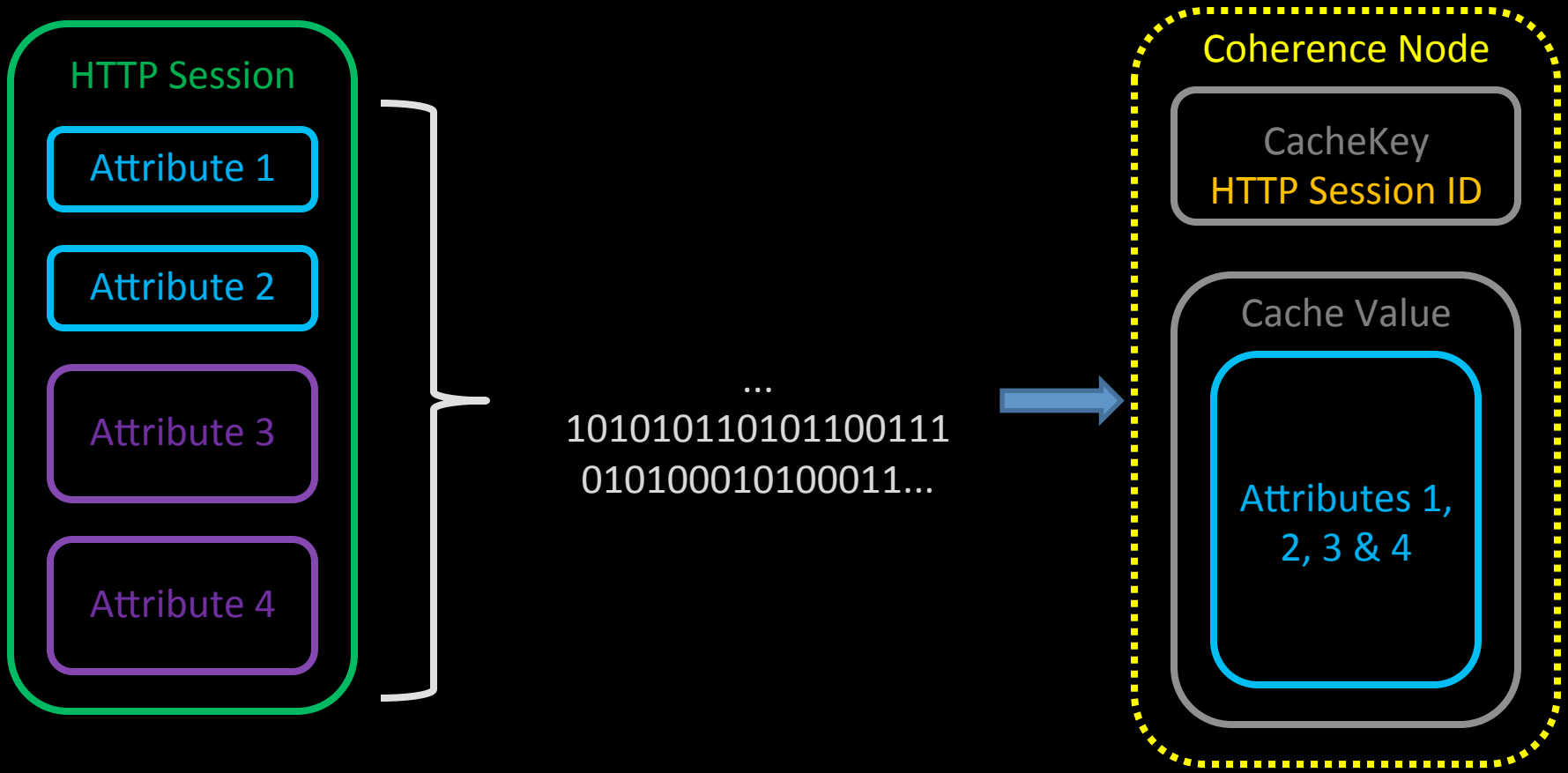


Traditional

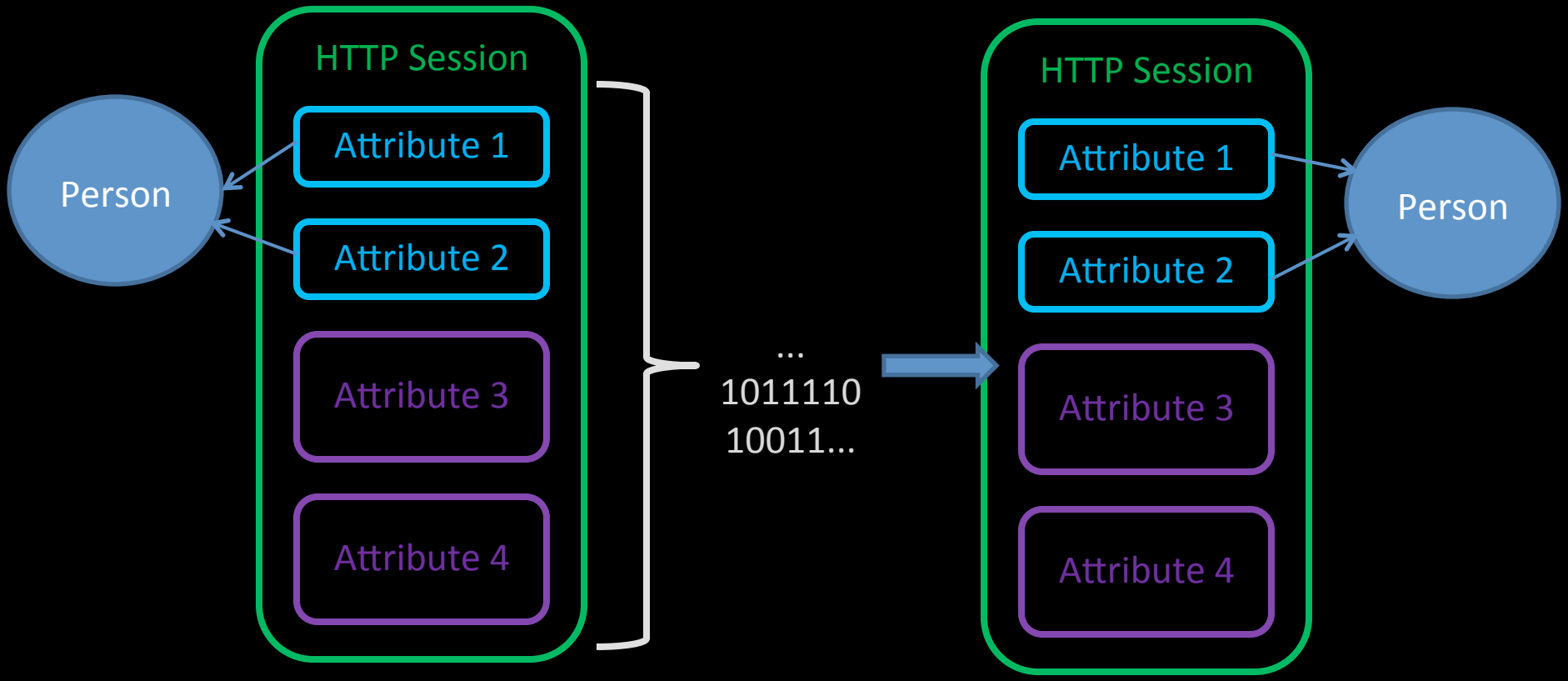
- Good for small Sessions
- Each Attribute is Serialized Independently
- All Attributes are transferred together
- Doesn’t work when attribute values point at the same Object

mon·o·lith·ic

Monolithic


HTTP Session
Attribute 1
Attribute 2
Attribute 3
Attribute 4
...
10101011010110011101010010100011...
Coherence Node
CacheKey
HTTP Session ID
Cache Value
Attributes 1, 2, 3 & 4

Monolithic

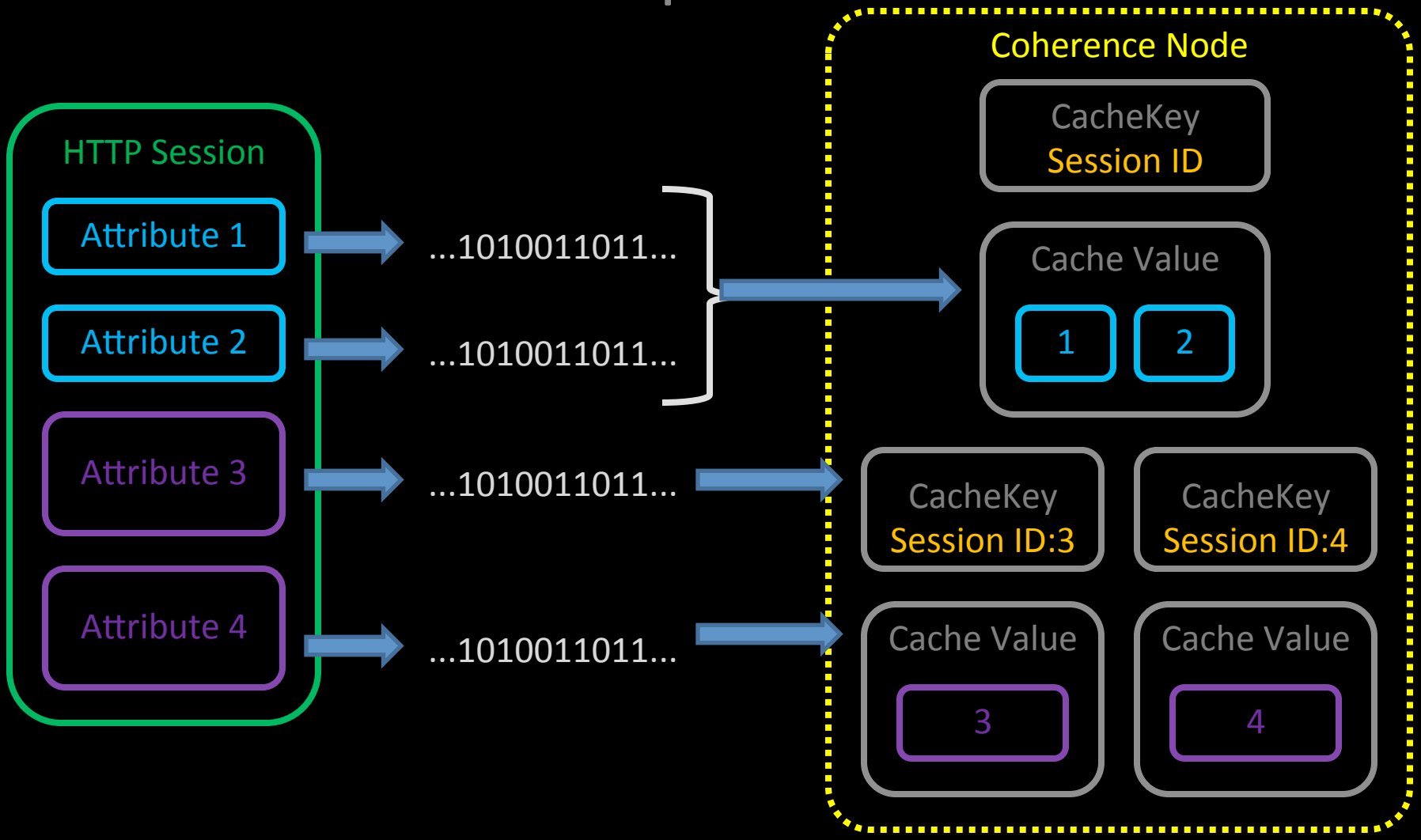

HTTP Session
Attribute 1
Attribute 2
Attribute 3
Attribute 4
Person
...
1011110
10011...
HTTP Session
Attribute 1
Attribute 2
Attribute 3
Attribute 4
Person

Monolithic

- Good when your Session attributes require shared Object references to be preserved
- Attributes are Serialized together
- All Attributes are transferred together
- Expensive – entire Session must be Serialized into a single Object Stream

Split

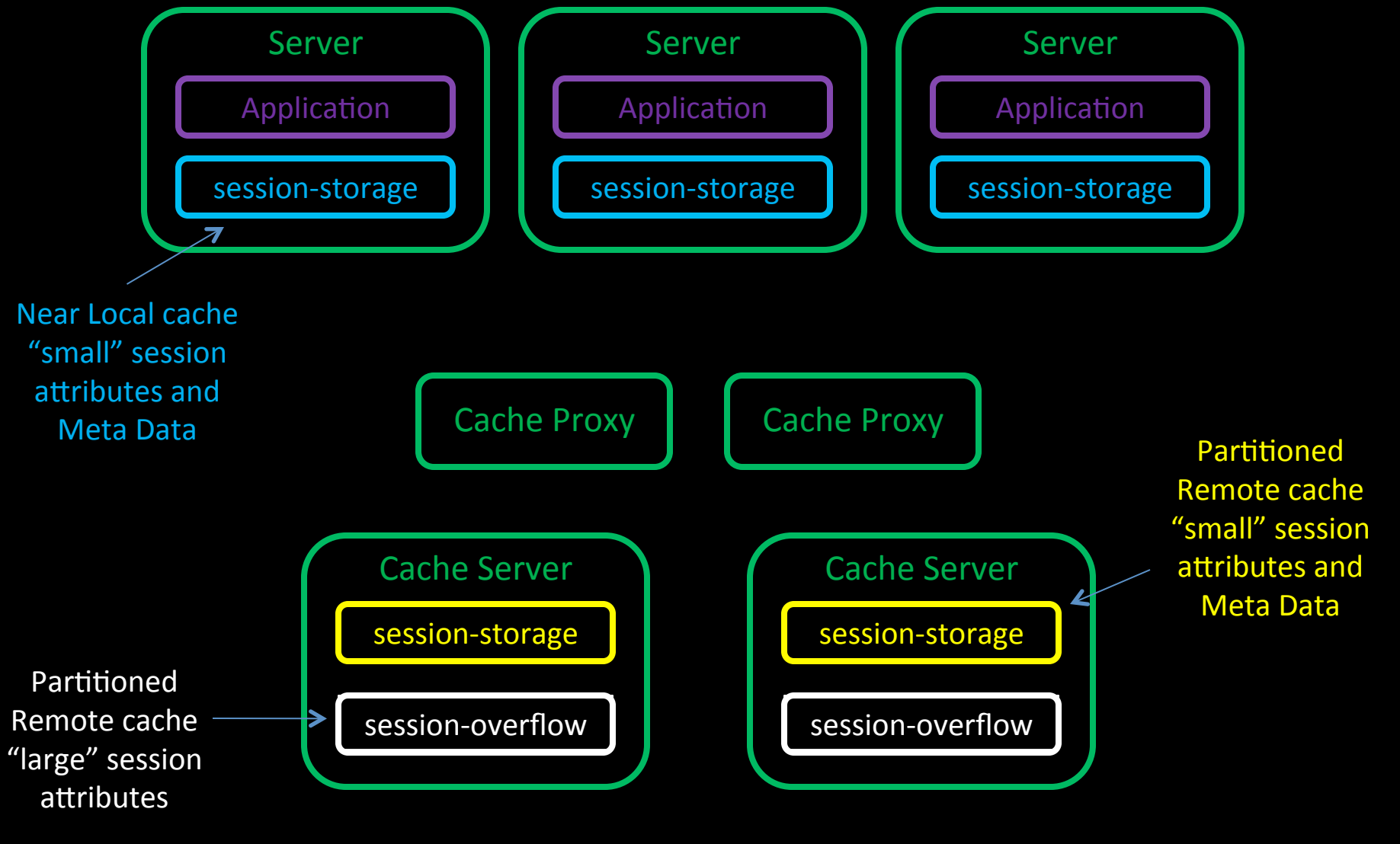
Split


Coherence Node
HTTP Session
Attribute 1
Attribute 2
Attribute 3
Attribute 4
...1010011011...
...1010011011...
...1010011011...
...1010011011...
CacheKey
Session ID
Cache Value
1
2
CacheKey
Session ID:3
CacheKey
Session ID:4
Cache Value
3
Cache Value
4

Split

- Default Coherence*Web Session Model
- Attributes are Serialized independently
- Large Attributes are transferred independently
- Reduces network traffic
- Separated handling of small and large attributes
- Configurable “Large” Attribute Threshold

Split in Action



Split

- Large Session Attributes are off-loaded to a remote cache scheme
- Frequently accessed attributes and MetaData held in near cache for performance

Tracking Attribute Changes

- Initial binary value vs New Binary Value
  - Object is mutable?
  - Initial == new (no change)
  - Initial != new (write to cache)
- Check can be overridden if **you** never mutate attributes without calling HttpSession.setAttribute
  - Removing the check improves performance

```
<context-param>
      <param-name>coherence-enable-suspect-attributes</param-name>
      <param-value>false</param-value>
</context-param>
```

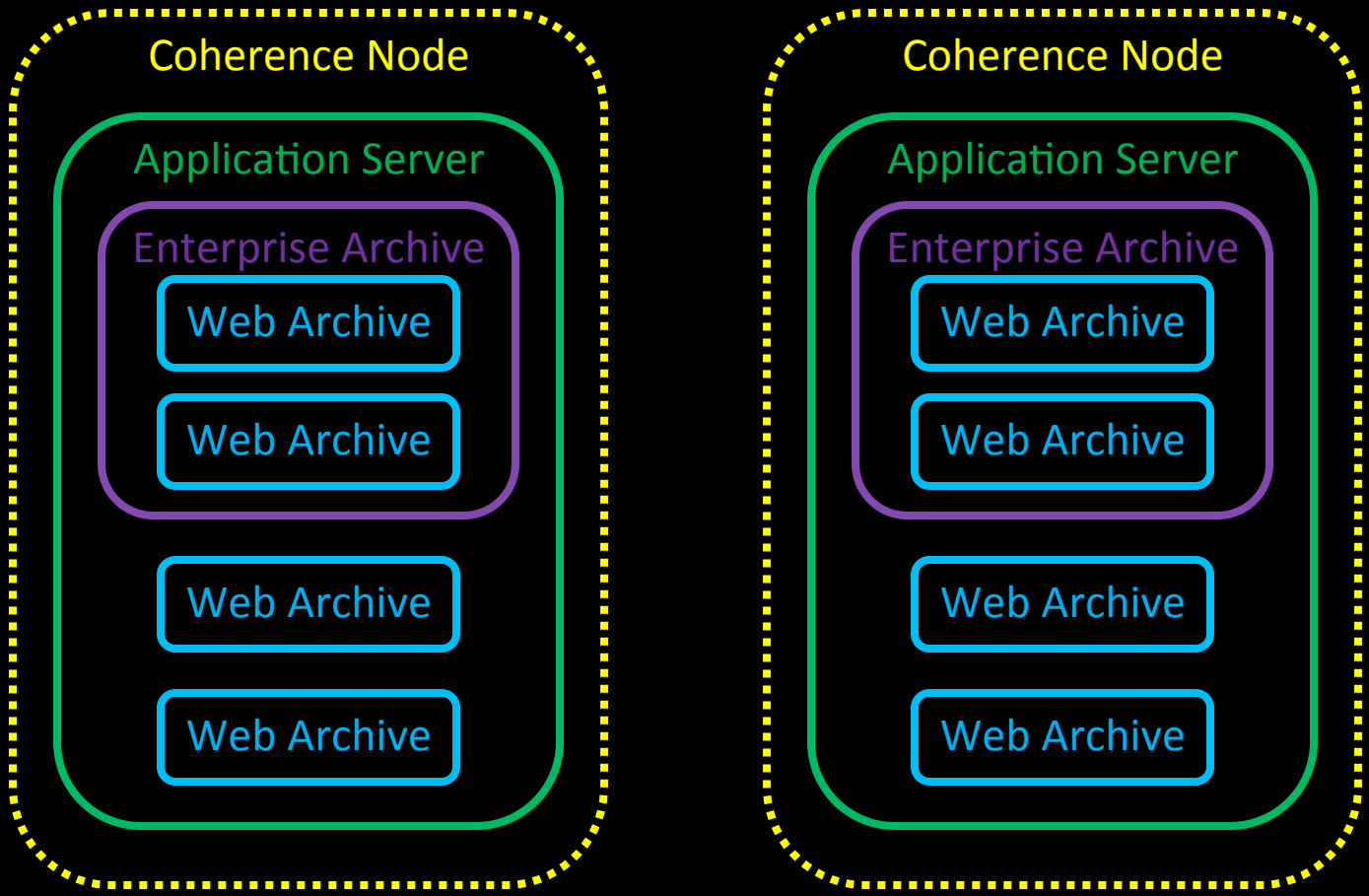
Cluster Isolation

Cluster Isolation Scope

- Container
- Enterprise Archive
- Web Archive

Determined by Class-loading Scope

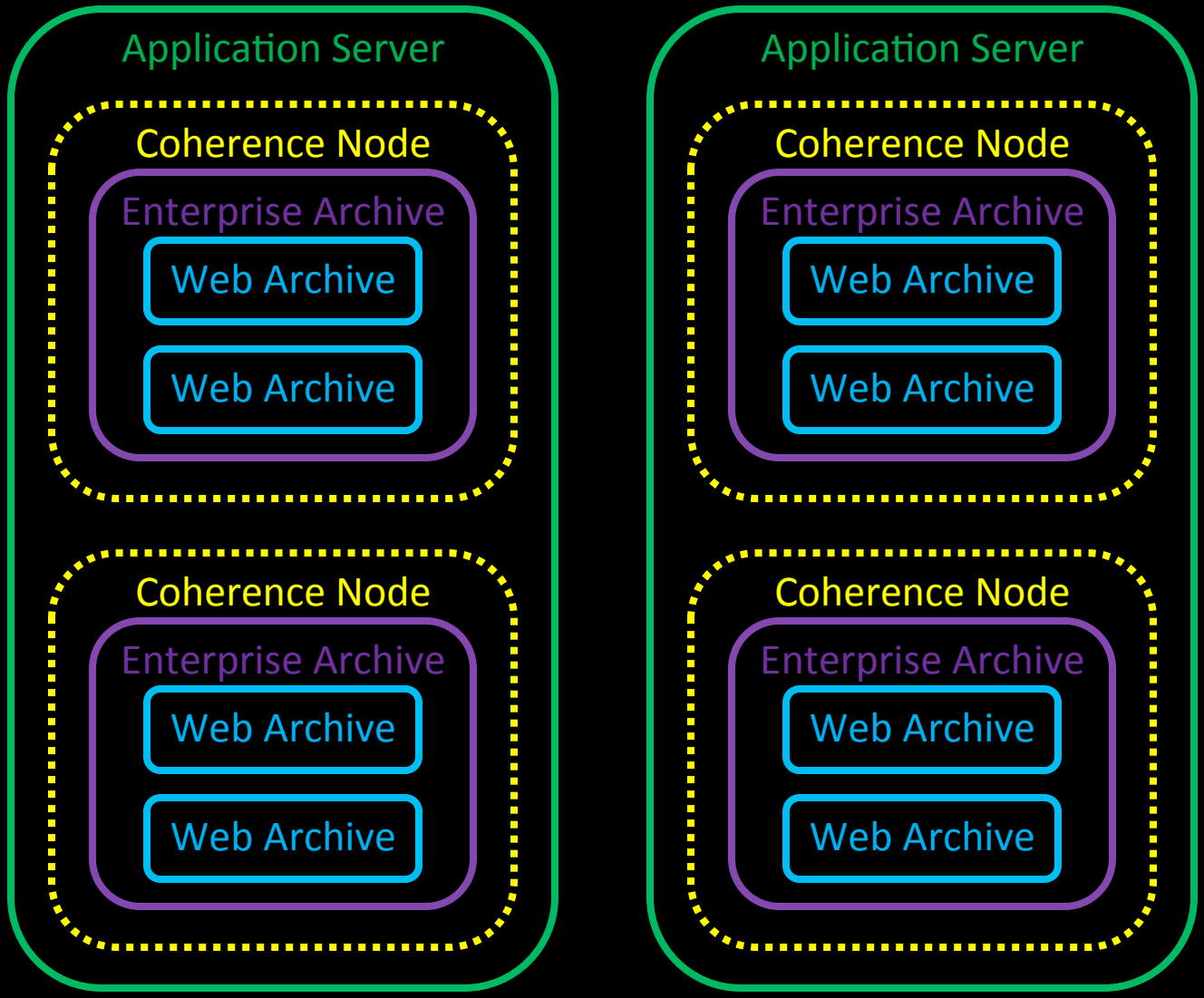
Container Scoping



Container Scoping

- Least Overhead
  - Libraries deployed on Container Class Path
  - Number of loaded Classes
  - Lowest Memory Requirement
  - One Coherence Cluster Node per Container
- Least Separation
  - Multiple Applications share the same Cluster
  - Can be problematic...

Enterprise Archive Scoping

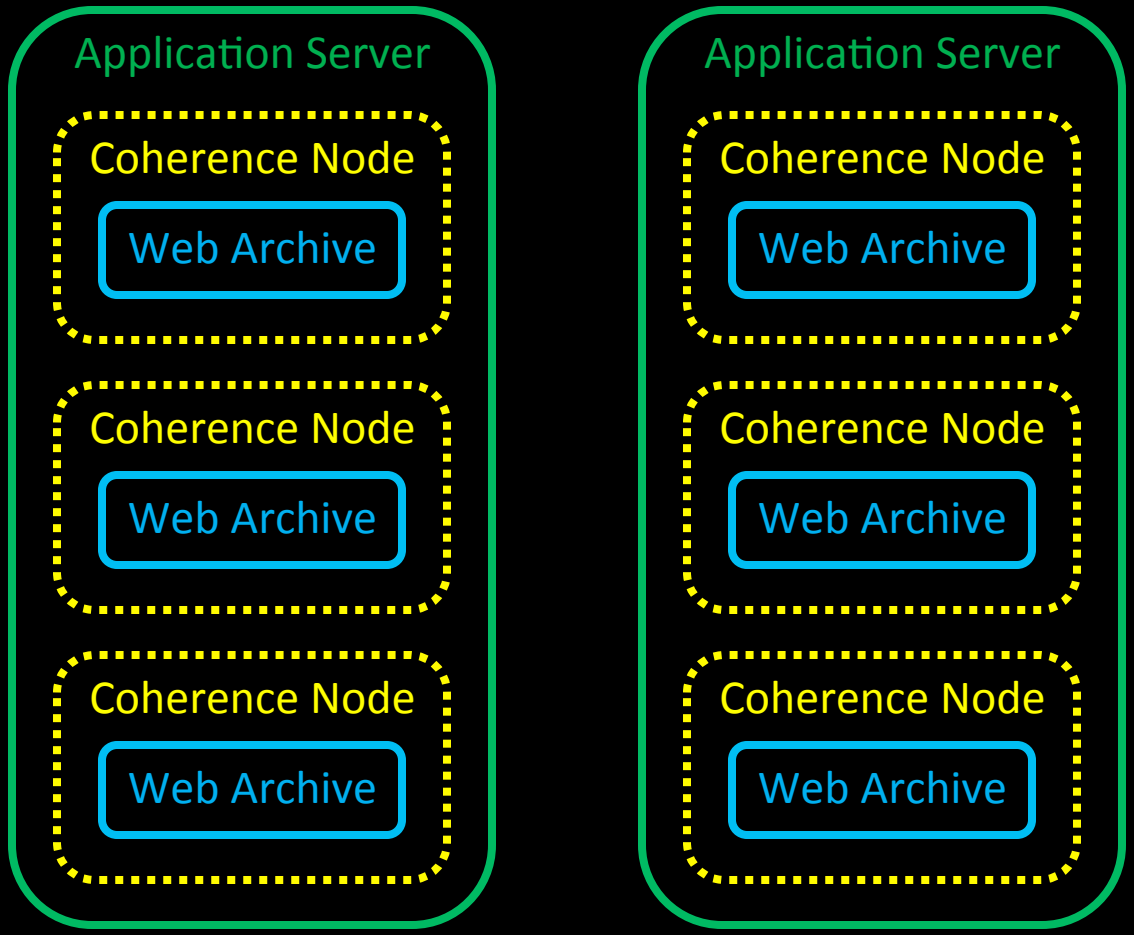


Enterprise Archive Scoping

- Overhead
  - Libraries scoped to EAR
  - One Coherence Cluster Node per Application
- Separation
  - Applications within the EAR share the same Cluster

```
<weblogic-application>
...
  <library-ref>
    <library-name>coherence</library-name>
  </library-ref>
    ...
  <library-ref>
    <library-name>active-cache</library-name>
  </library-ref>
...
</weblogic-application>
```

Web Archive Scoping



Web Archive Scoping

- Overhead
  - Libraries scoped to WAR
  - One Coherence Cluster Node per Web Archive
- Separation
  - Each Web Archive participates in its own cluster
  - Total Independence from negative external factors

Reference via MANIFEST.MF

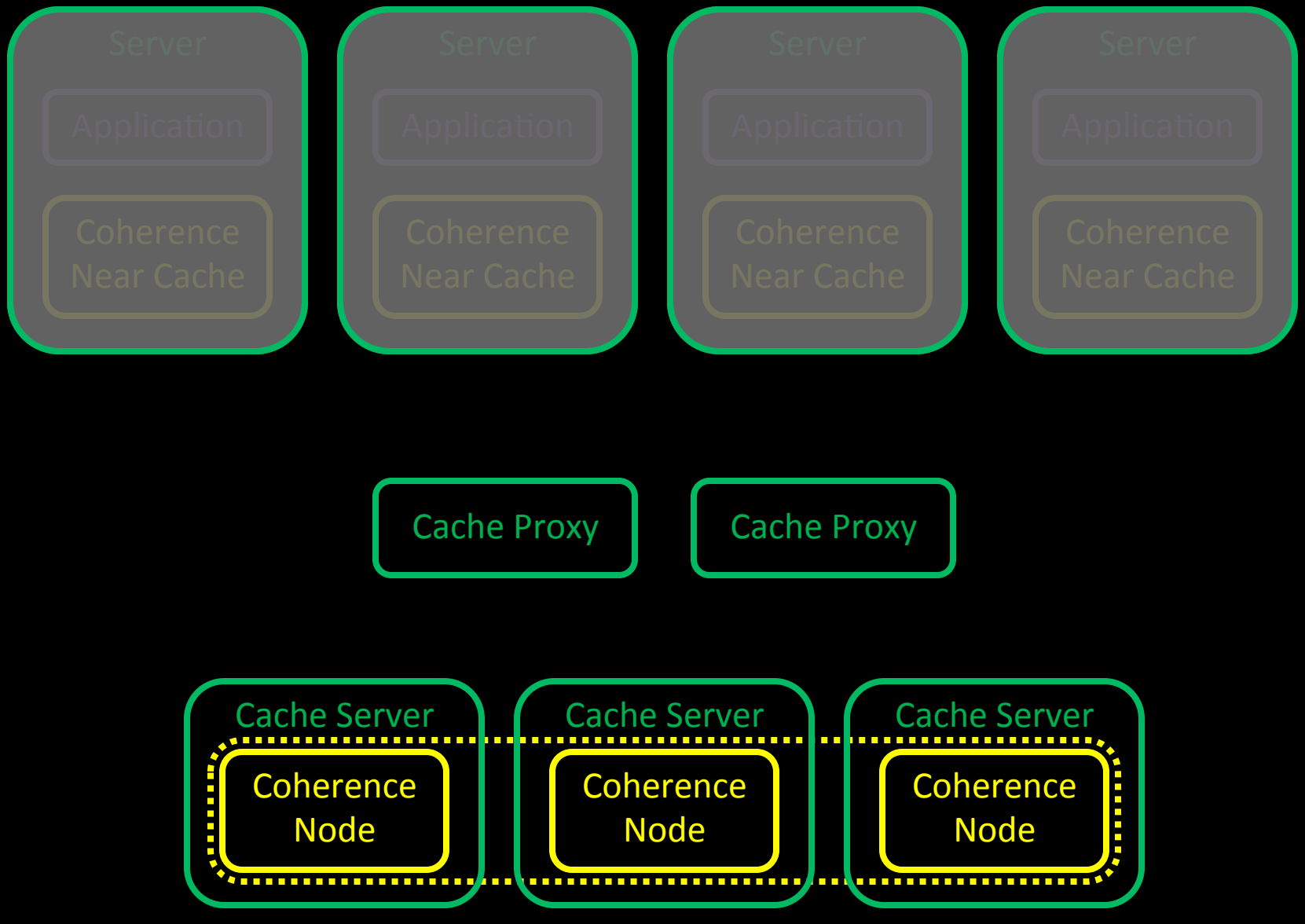
```
Manifest-Version: 1.0
Extension-List: coherence active-cache
coherence-Extension-Name: coherence
active-cache-Extension-Name: active-cache
```

...or package in WEB-INF/lib

So what can we do?

Application Tier Shutdown

Application Tier Shutdown

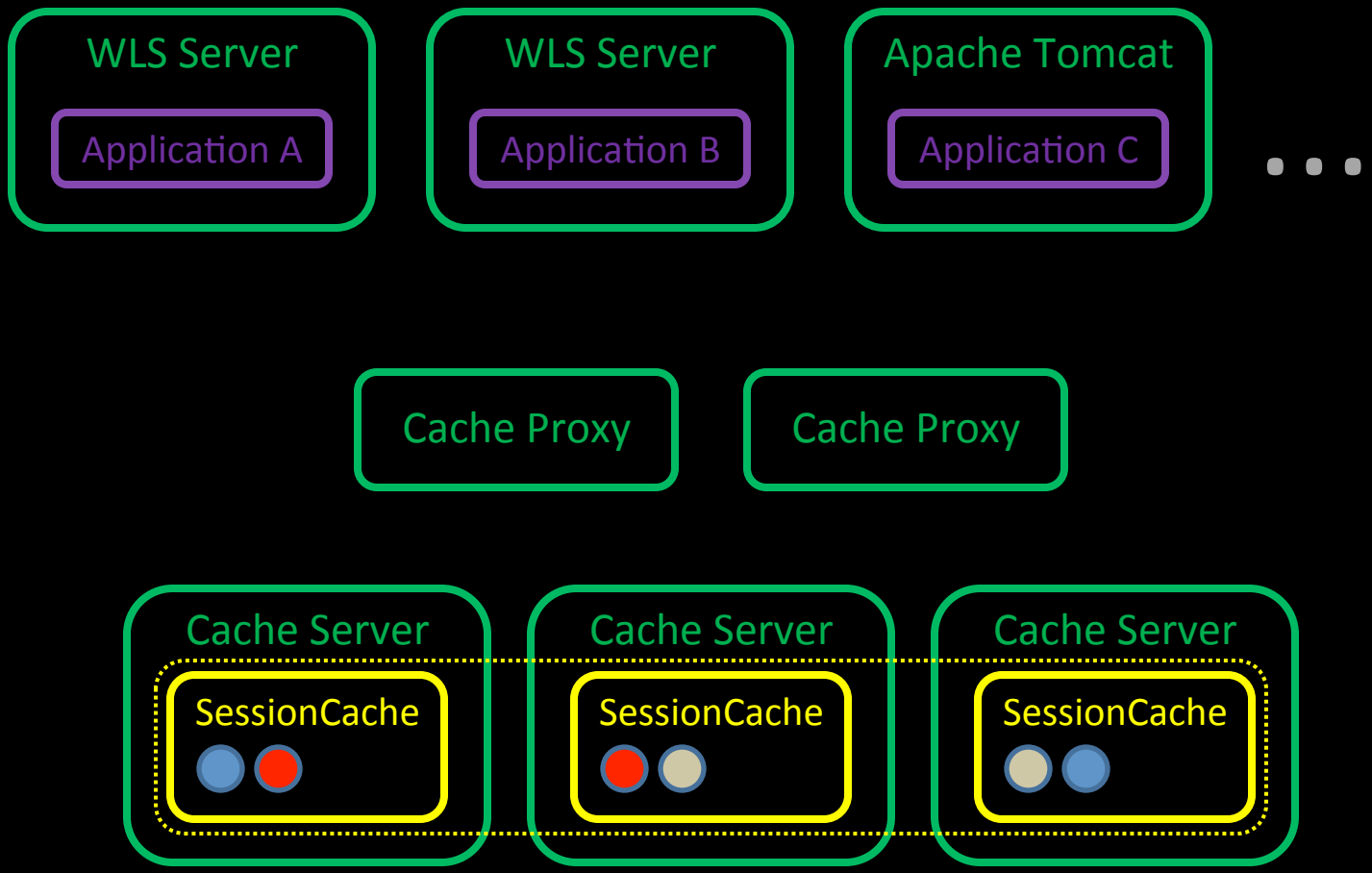


Application Tier Shutdown

- Application and Cache are totally separated
  - Session State is retained by the Cache Tier

Sharing Sessions

Sharing Session State Across Applications & Containers



Session & Attribute Scoping

Session Scoping

- For Applications running under separate domains without a load-balancer:
  - http://consulting.c2b2.com
  - http://marketing.c2b2.com

```
<context-param>
      <param-name>coherence-session-cookie-domain</param-name>
      <param-value>.c2b2.com</param-value>
</context-param>
```

- For Applications with different context paths:
  - http://www.acme.com/shared/portal
  - http://www.acme.com/shared/trading

```
<context-param>
      <param-name>coherence-session-cookie-path</param-name>
      <param-value>/shared</param-value>
</context-param>
```

Session Attribute Scoping

```
<context-param>
      <param-name>coherence-scopecontroller-class</param-name>
      <param-value>...</param-value>
</context-param>
```

- Global Scoping
  - AbstractHttpSessionCollection$GlobalScopeController
- Application Scoping
  - AbstractHttpSessionCollection$ApplicationScopeController
  - Allows different applications sharing HttpSessions to use identically named attributes in isolation
  - Default implementation applies a configurable prefix
  - Pluggable control over whether attributes are shared or isolated

```
<context-param>
      <param-name>coherence-application-name</param-name>
      <param-value>session-attribute-prefix</param-value>
</context-param>
```

Demo

Summary

- Independent Scaling
- Independent Tuning
- Near Cache Optimization offsets remote Latency
- Split Session Model Attribute separation
- Redundancy
- Scalability
- Availability
- Performance

Thanks for Listening
Any Questions?
http://www.c2b2.co.uk
http://blog.c2b2.co.uk
@c2b2consulting